Item - 6-

Home

3. Internet Radio

Broadcast radio hasn't changed much over the last few decades—it's always been a one-way medium. The program manager (and occasionally the DJ) determines the type of programming, including which songs are played and how often they are played. For the listener, it's a "take it or leave it" proposition. If you like what a station is playing, you listen to it; if you don't like it, then you either have to suffer through it or turn the dial until you find something better.

Because traditional radio is not an interactive medium, there is little feedback from listeners. In theory, listeners ultimately determine what music a station plays and how often. But listener feedback is indirect and slow through the existing rating systems. And the ratings systems are driven more by business considerations than by the preferences of individual listeners.

A common complaint about broadcast radio is that stations do little to help listeners identify songs. (How often have you heard a song on the radio and wanted to know the name of the song or artist, but the DJ never announced either one?) Other complaints include excessive amounts of commercials and a limited number of local stations to choose from.

Internet radio eliminates many of the shortcomings of broadcast radio because it's delivered through the Web—an inherently interactive medium. Internet radio offers dozens of stations per site and allows interactive feedback so each listener can directly influence programming.

Internet radio also gives you access to a far wider variety of stations and programming than traditional broadcast radio. Radio sites on the Web can have dozens of stations featuring uninterrupted music, comedy, sports and talk shows, news, special events and many other types of programming.

Internet radio isn't limited by geography like broadcast radio is. In fact, Internet radio is often used to extend the reach of regular broadcast stations. If you're traveling out of the broadcast area of your favorite home station, you may still be able to listen to it if they also transmit their programming over the Internet.

Another advantage of Internet radio over broadcast radio is its availability in buildings where radio reception is poor or regular radio isn't an option. As long as you have an Internet connection, you can tune in and listen anytime.

Most Internet stations display the name of the song and the artist the entire time the song is playing, and many stations can also display album graphics, credits, and lyrics, along with links to the artist's Web site. If you hear a song you like, many stations provide a link so you can purchase the song or album on the spot.

Some sites, like Imagine Radio, even allow you to set up a personal radio station, which you customize by selecting the artists and the types of music you want to hear. Once your radio station is set up, you can tune in and listen to music customized to your tastes. You can also make your station available to other listeners.

Major players like America Online and Rolling Stone Magazine are getting involved in Internet radio. AOL's Spinner.com Web site offers over 100 stations and a selection of more than 150,000 songs. Rolling Stone Radio features stations that play music selected by rock stars, such as David Bowie, and other celebrities. Many of these larger sites also offer music charts, industry news and other types of music-related content.

Many broadcast radio stations now have Web sites, and a few are beginning to offer their regular programming via the Internet. Sites such as Broadcast.com act as aggregators (collectors and distributors) of streaming media programming and Web content for both traditional radio stations and Internet-only radio stations.

Internet radio sites can generate advertising revenue with both announcement-type ads and banner ads. In this respect, the economic model of Internet radio is similar to broadcast radio. But Internet radio sites can expand on this model to earn commissions on products sold through their sites. Some sites even offer premium

subscription services, similar to cable and satellite TV.

Most of the larger Internet radio sites and services feature banner ads, but at least most of them avoid promotional announcements that interrupt the music. Some services, like vTuner, offer commercial-free listening if you purchase their "plus" software. Only a handful of the larger sites, such as Green Witch, remain commercial-free.

Due to the requirement for an Internet connection, Internet radio isn't yet as portable as broadcast radio. But within the next few years, hand-held PCs will be able to double as portable radios. (Hand-held PCs already offer wireless Internet access, and several models, such as the Cassiopeia E100, include sound capability and software to play MP3s.)

Sound Quality

The main factor limiting Internet radio is bandwidth. Dual channel ISDN (128 kbps) is the minimum needed for high-quality stereo music, but the majority of users have much slower connections. Voice quality is usually fine at slower connection speeds, but music quality is barely acceptable with connections slower than 56 kbps.

Other fast Internet connections such as DSL, cable modems, and satellite links provide enough bandwidth for CD-quality audio. However, even with unlimited bandwidth, network congestion can cause problems during peak usage periods. These problems will eventually be solved, but it could be years before the majority of Internet users have access to fast connections.

Another problem, even bigger than the connection speed of individual users, is that most streaming audio (and video) on the Internet is transmitted in a unicast mode—which is extremely inefficient. With unicast, each listener (or viewer) receives a separate stream. A station that has 500 users connected will send 500 copies of the same stream.

Even if all users had fast Internet connections, the Internet currently could handle only a few million simultaneous listeners with unicast transmissions. There is nowhere near enough server capacity and bandwidth to support tens of millions of listeners or viewers like network radio and network television can.

Eventually, the Internet will become multicast enabled, and a single stream will be able to be shared by multiple users. Only then will Internet radio be able to compete on the scale of traditional broadcast media. By the time that point is reached, the traditional radio and television networks will have had a chance to transition much of their programming to Internet.

Digital Radio

Digital audio technologies like MP3 are a key part of Internet radio because they help squeeze more sound through slower Internet connections. But digital technology can also be used in other forms of radio as well to improve the sound quality and transmit related information along with the music. Broadcast networks already use MP2 (similar to MP3) to transmit audio signals to their affiliate stations.

Eventually, all forms of radio will go digital. Cable and satellite TV systems already offer multiple music channels and have the capability to display text and video with music. Some systems, such as DMX, already offer digital transmission over cable and may eventually offer true interactivity.

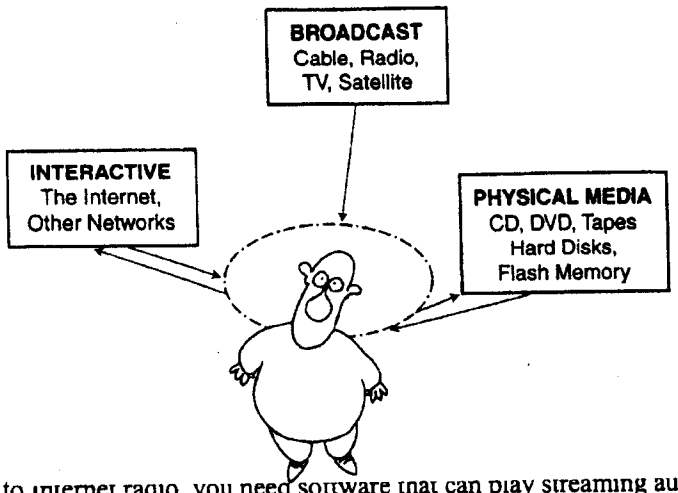
Traditional broadcast radio stations can use digital transmission to offer improved sound quality and display song titles and artist names along with the audio. Portable digital radios featuring a small display to show text and graphics will become commonplace within the next few years, and portable satellite and cellular radio services that allow a station to "follow" you as you drive across the continent will also become available.

Internet radio is available now, and, thanks to the high degree of interactivity provided by the Web, it opens the door to a whole new world that broadcast, cable and satellite radio can't. Eventually cable and satellite services will offer interactivity or even merge with Web TV and Internet radio. Until these media converge, consumers will be faced with a bewildering array of delivery mechanisms for audio and video content.

RN-00334

Listening to Internet Radio

Figure 4 - Media Overload



To listen to internet radio, you need software that can play streaming audio. Streaming audio is a subset of streaming media (audio, video and text, etc.) and comes in several formats, so you may need to install more than one program.

Many players, such as the RealPlayer G2 and the Windows Media Player, support multiple formats, including streaming MP3. The fact that there are multiple formats and players for streaming audio can be confusing. Fortunately, most sites include links for you to download any software required to listen to them.

At the very least, you should install the latest versions of the RealPlayer, Windows Media Player and at least one of the full-featured MP3 players, such as Sonique or Winamp. These programs will allow you to listen to the majority of Internet radio sites.

Most players can be downloaded for free, although there are a few, such as the Plus version of the RealPlayer, that you must purchase. Fortunately, the free version of the RealPlayer offers everything most users need. The Windows Media Player is included with Windows. Updates can be obtained for free directly from Microsoft. Sonique and Winamp are both freeware.

Many of the larger radio sites, such as Spinner.com and Rolling Stone Radio, require you to install their own "tuners." Both of these sites use RealAudio, and you'll need to install the RealPlayer along with their own software. Many of these larger sites also require you to register before you can download their software, and some require you to login each time you listen.

vTuner

vTuner Main Screen

vTuner (www.vtuner.com) is a program, based on the Real player, which provides an easy way to find and listen to thousands of stations (radio, television, Webcam, and others) from all over the world. The free version of vTuner categorizes stations by type and geographic location and provides browsing and searching capabilities. vTuner Plus ($29.99) adds station scanning, playback scheduling, station ratings based on quality and reliability, and replaceable "skins." The Plus version also lets you avoid listening to advertisements.

Table 2 lists some of the more popular streaming media player software that can be used to listen to Internet radio.

Table 1 - Streaming Media Players

Player	Streaming Formats	Web Site
QuickTime	QuickTime, MP3 and others	www.quicktime.com
RealPlayer	RealAudio, MP3 and others	www.realaudio.com
Rolling Stone Radio Player	RealAudio	www.rsradio.com
Spinner	RealAudio	www.spinner.com
vTuner	RealAudio	www.vtuner.com
Winamp	MP3 and others	www.winamp.com
Winplay	Encrypted MP3	www.radiomoi.com
Windows Media Player	WMA, MP3 and others	www.microsoft.com/windows/windowsmedia

Popular Radio Web Sites

A sampling of popular Internet radio sites follows. Many more sites are available but aren't covered here due to space limitations. (For more listings, see the Internet Radio section of Appendix A, *Interesting Web Sites*.)

Green Witch

Green Witch (*www.greenwitch.com*) is an easy-to-use site that offers a wide range of commercial-free music on multiple channels, including genres such as alternative rock, blues, classical, hip-hop and more.

Green Witch helps link independent artists to fans. Artists can have their songs added to Green Witch's streams with features to help them sell music, including an artist information page, a "Buy" button that links to retail fulfillment and a "Download" button for on-demand retail.

Green Witch also provides links to dozens of independent Icecast stations with offerings that range from various genres of music to comedy and talk shows with offerings such as Rush Limbaugh and animal noises. (See if you can tell the difference between the latter two.)

The channels offered by Green Witch use streaming MP3. To listen to a channel, click on the speaker icon to the left of the channel name.

Imagine Radio

At Imagine Radio (*www.imagineradio.com*) you can listen to music from the site's own stations, listen to other people's customized radio stations, or even create your own personal radio station. You can also buy a CD on the spot if you like a song after listening to it.

To listen to music, choose a station and click on the **Listen** button. The Imagine Radio tuner will load and, after a few seconds, a song will start playing. The tuner will then display the title and artist name. To stop a song that's playing, click on the **Pause** button above the song title. You can skip forward to the next song, but you can't go back to the previous song because of current webcasting laws.

To create your own customized radio station, you first select a name, a "scene," and the music genres (Blues, Jazz, Rock, etc.) to include. Then you browse a list of artists and rank them depending on how frequently you want their songs to be played. To listen to your station, select the **play my station** button from the main page.

You can rate any song by clicking on the **Edit** button while the song is playing. This influences how often the song plays on your station in the future. Ratings go from 0 to 5 and IR. If you hate the song and never want to hear it played on your station, select **0**. If you love the song and want to hear it frequently, select **5**. To have the Imagine Radio DJ decide how often the song is played, select **IR**.

You can make your custom station available to others, and you can even e-mail friends a link to your station. You can also listen to radio stations that other listeners have put together, (with names like Fartsniffer and RaveZone) grouped in such themes as Carnival, French Quarter and Woodstock.

To listen to Imagine Radio, you need either the RealPlayer G2 or the Windows Media Player.

RadioMoi

RadioMoi (*www.radiomoi.com*) provides access to thousands of songs, an Interactive Music Library that lets you play DJ and create your own shows, and an Interactive Jukebox that lets you select songs to be played on demand (only a portion of the songs are approved for interactive access). Channels (RadioMoi calls them shows) include an array of music, comedy and celebrity interviews.

RadioMoi was the first webcaster to sign an agreement with the RIAA and to be licensed under the Digital Millennium Copyright Act. This license allows RadioMoi to stream copyrighted sound recordings and requires them to make royalty payments. RadioMoi also provides links to artist and record label Web sites and lets listeners purchase albums on the spot.

Radio Moi uses an encrypted form of MP3. To listen to audio, you must use Winplay (the free RadioMoi player). If you have other MP3 players, such as Winamp, installed you may need to change the application associated with the .M3U file type. Otherwise, your MP3 player may attempt to play the RadioMoi channel—which it can't because of the encryption. (See Chapter 9, *Organizing and Playing Music*, for more information on file type associations.)

Rolling Stone Radio

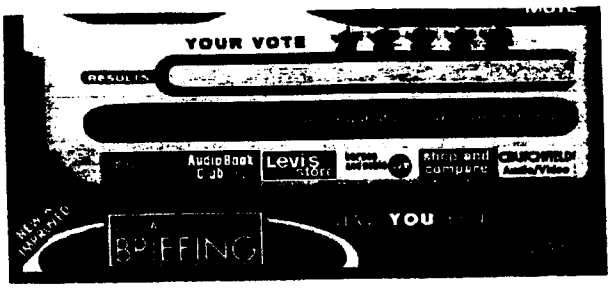
Rolling Stone Radio (*www.rsradio.com*) is Rolling Stone Magazine's Internet radio site. It offers a diverse selection of music channels, including DBRN, which is the David Bowie Radio Network, offering the rock star's favorite music.

Rolling Stone Radio's channels are located on the left side of the player. Only some of them show, so you'll need to use the up and down arrow keys on the player to scroll through them. Once you find a channel you want to listen to, click the **Play** button. After several seconds, the song will begin to play, and the title and artist name will be displayed. Click on the artist name to see more information about that artist.

You can rate any song while it's playing by clicking on one of the checkboxes labeled 1-5. If you like the music, you can click on a link that will take you to Amazon.com, where you can purchase the album. You can also click on a button to submit a song request to the Rolling Stone Radio DJ.

To play Rolling Stone Radio, you need the RealPlayer G2 and the Rolling Stone Radio tuner, both of which can be downloaded at the rsradio.com site for free.

Rolling Stone Radio Tuner



Spinner

Spinner (*www.spinner.com*) is owned by America Online and offers access to more than 150,000 songs across 100+ music channels, grouped by genre, with programmable presets. You can rate any song that's played, access artist information, and, if you want, purchase the CD. Currently, Spinner doesn't allow you to set up your own station.

To listen to Spinner on a Windows system, you need to install their stand-alone player. If you're on a Mac or Unix system, Spinner offers a player that runs in conjunction with your Web browser. You'll also need to install the *RealPlayer*.

Streaming MP3

Streaming MP3 has rapidly become the choice for amateur webcasters worldwide. Now, anyone with a PC and an Internet connection can inexpensively stream music to listeners throughout the world, using SHOUTcast or Icecast streaming MP3 software.

Spinner Plus Tuner

SHOUTcast

Nullsoft's SHOUTcast (*www.shoutcast.com*) provides users with a simple way to stream MP3 music to listeners all over the world. SHOUTcast servers can submit their description and status back to the main SHOUTcast server directory, which allows listeners to locate SHOUTcast servers without knowing their IP addresses.

Icecast

Icecast (*www.icecast.org*) is an open source streaming MP3 server, similar to SHOUTcast. It is available for free, including the source code. Because it is open source, useful modifications and additions by users are incorporated back into the main code for the benefit of all users.

MP3Spy

MP3Spy (*www.mp3spy.com*) helps you find SHOUTcast servers and listen to webcasters from all over the world. MP3Spy lists the available SHOUTcast servers and identifies them by music genre and type of programming.

When you choose a server, MP3Spy connects you to the server's audio stream and the Web page of the station. When you connect to a SHOUTcast server, you can also chat with the DJ or other users who are listening to the same music. You'll need an MP3 player, such as Winamp or Sonique, to use MP3Spy.

Webcasting Licensing

Internet radio stations can give listeners a high degree of control over the music they hear, but the music industry seems to fear that giving listeners too much control will reduce music sales. Their reasoning seems to be that if people could choose to listen to any song at any time, there would be little incentive for anyone to actually purchase music.

While the recording industry was slow to recognize the potential of downloadable music, it was quicker to recognize the potential (and threat) of Internet radio and lobbied to have laws enacted to protect its interests. The Digital Millennium Copyright Act (sponsored by the recording industry) addresses the issue of webcasting by providing statutory (mandated by law) licenses for webcasters who meet certain conditions. (See Chapter 5, *Digital Music and Copyright Law*, for the requirements for statutory webcasting licenses.)

Some Internet radio sites exist that webcast music illegally, but many webcasters want to be "legal" and are obtaining or have obtained the licensing required. Amateur webcasters are popularizing streaming audio, just like grass roots support and the Internet popularized MP3. But the recording industry is bent on ensuring that proper royalties are paid whenever copyrighted music is played and that music streamed over the Internet doesn't cut in to music sales.

In addition to licensing fees, webcasters are subject to several significant restrictions. For example, while Internet radio listeners can select the songs they want to hear, it is illegal for webcasters to allow them to select a particular song to play instantly, unless the song has been specifically authorized for interactive distribution. Even though listeners can create personalized stations, the site's DJ must rotate the playlists and determine when each song is played.

Webcasters are concerned that these types of restrictions will inhibit their ability to play the music that listeners want to hear, and make it financially unfeasible to operate a radio site. Internet radio is evolving rapidly, and more legislation may be required as it matures. Eventually, more standards and laws will be established and Internet radio will become a major component of our media, just like broadcast radio and television. Until then, it's still a bit like the Wild Wild West.

Item -7-

The MP3 and Internet Audio Handbook

Your Guide to the Digital Music Revolution!

ISBN# 1-928791-10-7 $24.95

Order here and help support your independent author

Digital / PDF - $9.95 (60% Savings)

Paperback - $19.95 (20% Savings)

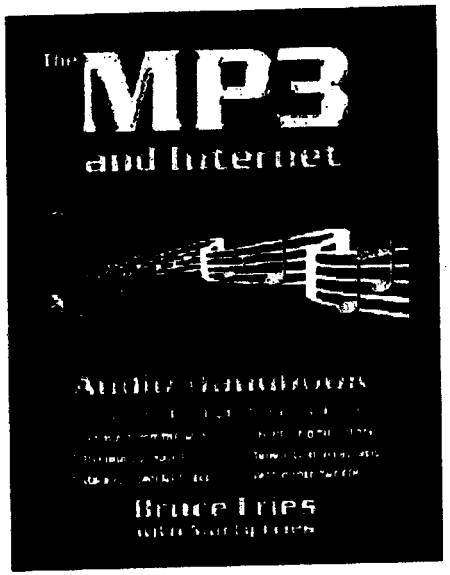
Listen to the author's station at [illegible]

"The bible of MP3...a great resource stuffed full of great advice and picture-driven tutorials; lovely stuff." ***Read full review***
- Paul Cooke, MusicDish.com
"One of the best overall discussions of how sound and digital audio technology works, anywhere." ***Read full review***
- Mike Powers, About.com Internet Radio Guide
"Highly recommended, especially for those new to computer digital music."
- The Midwest Book Review

The MP3 and Internet Audio Handbook cuts through the hype and confusion surrounding MP3 and other digital music technologies. This book explains the benefits digital audio compression, downloadable music and streaming audio in simple, easy-to-understand terms, and includes over 80 charts and illustrations, along with tutorials for the most popular digital music software.

Included are step-by-step instructions for finding music on the Internet, listening to Internet radio, creating and playing MP3 files, recording music on a computer, recording custom CDs, and connecting a PC to a stereo system. This book provides product recommendations, along with lists of popular Internet radio stations and downloadable music sites, and complete coverage of the issues related to copyright laws and digital music.

RN-00201



Visit the authors at one of their free MP3 seminars

Click here for press release

Click here for tour schedule

Click here for seminar outline

TeamCom Books • P.O. Box 1251 Burtonsville, MD 20866 • 301-847-7600 • Fax 301-847-7638

RN-00202

Home

3. Internet Radio

Broadcast radio hasn't changed much over the last few decades—it's always been a one-way medium. The program manager (and occasionally the DJ) determines the type of programming, including which songs are played and how often they are played. For the listener, it's a "take it or leave it" proposition. If you like what a station is playing, you listen to it; if you don't like it, then you either have to suffer through it or turn the dial until you find something better.

Because traditional radio is not an interactive medium, there is little feedback from listeners. In theory, listeners ultimately determine what music a station plays and how often. But listener feedback is indirect and slow through the existing rating systems. And the ratings systems are driven more by business considerations than by the preferences of individual listeners.

A common complaint about broadcast radio is that stations do little to help listeners identify songs. (How often have you heard a song on the radio and wanted to know the name of the song or artist, but the DJ never announced either one?) Other complaints include excessive amounts of commercials and a limited number of local stations to choose from.

Internet radio eliminates many of the shortcomings of broadcast radio because it's delivered through the Web—an inherently interactive medium. Internet radio offers dozens of stations per site and allows interactive feedback so each listener can directly influence programming.

Internet radio also gives you access to a far wider variety of stations and programming than traditional broadcast radio. Radio sites on the Web can have dozens of stations featuring uninterrupted music, comedy, sports and talk shows, news, special events and many other types of programming.

Internet radio isn't limited by geography like broadcast radio is. In fact, Internet radio is often used to extend the reach of regular broadcast stations. If you're traveling out of the broadcast area of your favorite home station, you may still be able to listen to it if they also transmit their programming over the Internet.

Another advantage of Internet radio over broadcast radio is its availability in buildings where radio reception is poor or regular radio isn't an option. As long as you have an Internet connection, you can tune in and listen anytime.

Most Internet stations display the name of the song and the artist the entire time the song is playing, and many stations can also display album graphics, credits, and lyrics, along with links to the artist's Web site. If you hear a song you like, many stations provide a link so you can purchase the song or album on the spot.

Some sites, like Imagine Radio, even allow you to set up a personal radio station, which you customize by selecting the artists and the types of music you want to hear. Once your radio station is set up, you can tune in and listen to music customized to your tastes. You can also make your station available to other listeners.

Major players like America Online and Rolling Stone Magazine are getting involved in Internet radio. AOL's Spinner.com Web site offers over 100 stations and a selection of more than 150,000 songs. Rolling Stone Radio features stations that play music selected by rock stars, such as David Bowie, and other celebrities. Many of these larger sites also offer music charts, industry news and other types of music-related content.

Many broadcast radio stations now have Web sites, and a few are beginning to offer their regular programming via the Internet. Sites such as Broadcast.com act as aggregators (collectors and distributors) of streaming media programming and Web content for both traditional radio stations and Internet-only radio stations.

Internet radio sites can generate advertising revenue with both announcement-type ads and banner ads. In this respect, the economic model of Internet radio is similar to broadcast radio. But Internet radio sites can expand on this model to earn commissions on products sold through their sites. Some sites even offer premium subscription services, similar to cable and satellite TV.

Most of the larger Internet radio sites and services feature banner ads, but at least most of them avoid promotional announcements that interrupt the music. Some services, like vTuner, offer commercial-free listening if you purchase their "plus" software. Only a handful of the larger sites, such as Green Witch, remain commercial-free.

Due to the requirement for an Internet connection, Internet radio isn't yet as portable as broadcast radio. But within the next few years, hand-held PCs will be able to double as portable radios. (Hand-held PCs already offer wireless Internet access, and several models, such as the Cassiopeia E100, include sound capability and software to play MP3s.)

Sound Quality

The main factor limiting Internet radio is bandwidth. Dual channel ISDN (128 kbps) is the minimum needed for high-quality stereo music, but the majority of users have much slower connections. Voice quality is usually fine at slower connection speeds, but music quality is barely acceptable with connections slower than 56 kbps.

Other fast Internet connections such as DSL, cable modems, and satellite links provide enough bandwidth for CD-quality audio. However, even with unlimited bandwidth, network congestion can cause problems during peak usage periods. These problems will eventually be solved, but it could be years before the majority of Internet users have access to fast connections.

Another problem, even bigger than the connection speed of individual users, is that most streaming audio (and video) on the Internet is transmitted in a unicast mode—which is extremely inefficient. With unicast, each listener (or viewer) receives a separate stream. A station that has 500 users connected will send 500 copies of the same stream.

Even if all users had fast Internet connections, the Internet currently could handle only a few million simultaneous listeners with unicast transmissions. There is nowhere near enough server capacity and bandwidth to support tens of millions of listeners or viewers like network radio and network television can.

Eventually, the Internet will become multicast enabled, and a single stream will be able to be shared by multiple users. Only then will Internet radio be able to compete on the scale of traditional broadcast media. By the time that point is reached, the traditional radio and television networks will have had a chance to transition much of their programming to Internet.

Digital Radio

Digital audio technologies like MP3 are a key part of Internet radio because they help squeeze more sound through slower Internet connections. But digital technology can also be used in other forms of radio as well to improve the sound quality and transmit related information along with the music. Broadcast networks already use MP2 (similar to MP3) to transmit audio signals to their affiliate stations.

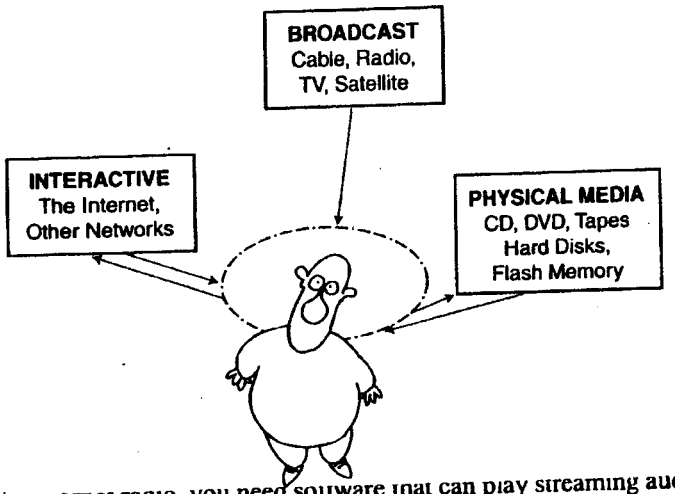
Eventually, all forms of radio will go digital. Cable and satellite TV systems already offer multiple music channels and have the capability to display text and video with music. Some systems, such as DMX, already offer digital transmission over cable and may eventually offer true interactivity.

Traditional broadcast radio stations can use digital transmission to offer improved sound quality and display song titles and artist names along with the audio. Portable digital radios featuring a small display to show text and graphics will become commonplace within the next few years, and portable satellite and cellular radio services that allow a station to "follow" you as you drive across the continent will also become available.

Internet radio is available now, and, thanks to the high degree of interactivity provided by the Web, it opens the door to a whole new world that broadcast, cable and satellite radio can't. Eventually cable and satellite services will offer interactivity or even merge with Web TV and Internet radio. Until these media converge, consumers will be faced with a bewildering array of delivery mechanisms for audio and video content.

Listening to Internet Radio

Figure 4 - Media Overload



To listen to internet radio, you need software that can play streaming audio. Streaming audio is a subset of streaming media (audio, video and text, etc.) and comes in several formats, so you may need to install more than one program.

Many players, such as the RealPlayer G2 and the Windows Media Player, support multiple formats, including streaming MP3. The fact that there are multiple formats and players for streaming audio can be confusing. Fortunately, most sites include links for you to download any software required to listen to them.

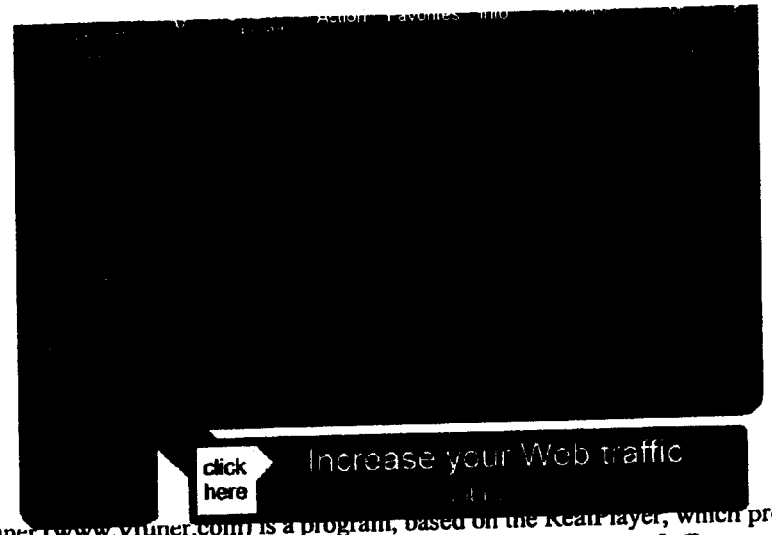
At the very least, you should install the latest versions of the RealPlayer, Windows Media Player and at least one of the full-featured MP3 players, such as Sonique or Winamp. These programs will allow you to listen to the majority of Internet radio sites.

Most players can be downloaded for free, although there are a few, such as the Plus version of the RealPlayer, that you must purchase. Fortunately, the free version of the RealPlayer offers everything most users need. The Windows Media Player is included with Windows. Updates can be obtained for free directly from Microsoft. Sonique and Winamp are both freeware.

Many of the larger radio sites, such as Spinner.com and Rolling Stone Radio, require you to install their own "tuners." Both of these sites use RealAudio, and you'll need to install the RealPlayer along with their own software. Many of these larger sites also require you to register before you can download their software, and some require you to login each time you listen.

vTuner

vTuner Main Screen



vTuner (www.vtuner.com) is a program, based on the Real Player, which provides an easy way to find and listen to thousands of stations (radio, television, Webcam, and others) from all over the world. The free version of vTuner categorizes stations by type and geographic location and provides browsing and searching capabilities. vTuner Plus ($29.99) adds station scanning, playback scheduling, station ratings based on quality and reliability, and replaceable "skins." The Plus version also lets you avoid listening to advertisements.

Table 2 lists some of the more popular streaming media player software that can be used to listen to Internet radio.

Table 1 - Streaming Media Players

Player	Streaming Formats	Web Site
QuickTime	QuickTime, MP3 and others	www.quicktime.com
RealPlayer	RealAudio, MP3 and others	www.realaudio.com
Rolling Stone Radio Player	RealAudio	www.rsradio.com
Spinner	RealAudio	www.spinner.com
vTuner	RealAudio	www.vtuner.com
Winamp	MP3 and others	www.winamp.com
Winplay	Encrypted MP3	www.radiomoi.com
Windows Media Player	WMA, MP3 and others	www.microsoft.com/windows/windowsmedia

RN-00207

Popular Radio Web Sites

A sampling of popular Internet radio sites follows. Many more sites are available but aren't covered here due to space limitations. (For more listings, see the Internet Radio section of Appendix A, *Interesting Web Sites*.)

Green Witch

Green Witch (*www.greenwitch.com*) is an easy-to-use site that offers a wide range of commercial-free music on multiple channels, including genres such as alternative rock, blues, classical, hip-hop and more.

Green Witch helps link independent artists to fans. Artists can have their songs added to Green Witch's streams with features to help them sell music, including an artist information page, a "Buy" button that links to retail fulfillment and a "Download" button for on-demand retail.

Green Witch also provides links to dozens of independent Icecast stations with offerings that range from various genres of music to comedy and talk shows with offerings such as Rush Limbaugh and animal noises. (See if you can tell the difference between the latter two.)

The channels offered by Green Witch use streaming MP3. To listen to a channel, click on the speaker icon to the left of the channel name.

Imagine Radio

At Imagine Radio (*www.imagineradio.com*) you can listen to music from the site's own stations, listen to other people's customized radio stations, or even create your own personal radio station. You can also buy a CD on the spot if you like a song after listening to it.

To listen to music, choose a station and click on the **Listen** button. The Imagine Radio tuner will load and, after a few seconds, a song will start playing. The tuner will then display the title and artist name. To stop a song that's playing, click on the **Pause** button above the song title. You can skip forward to the next song, but you can't go back to the previous song because of current webcasting laws.

To create your own customized radio station, you first select a name, a "scene," and the music genres (Blues, Jazz, Rock, etc.) to include. Then you browse a list of artists and rank them depending on how frequently you want their songs to be played. To listen to your station, select the **play my station** button from the main page.

You can rate any song by clicking on the **Edit** button while the song is playing. This influences how often the song plays on your station in the future. Ratings go from 0 to 5 and IR. If you hate the song and never want to hear it played on your station, select **0**. If you love the song and want to hear it frequently, select **5**. To have the Imagine Radio DJ decide how often the song is played, select **IR**.

You can make your custom station available to others, and you can even e-mail friends a link to your station. You can also listen to radio stations that other listeners have put together, (with names like Fartsniffer and RaveZone) grouped in such themes as Carnival, French Quarter and Woodstock.

To listen to Imagine Radio, you need either the RealPlayer G2 or the Windows Media Player.

RadioMoi

RadioMoi (*www.radiomoi.com*) provides access to thousands of songs, an Interactive Music Library that lets you play DJ and create your own shows, and an Interactive Jukebox that lets you select songs to be played on demand (only a portion of the songs are approved for interactive access). Channels (RadioMoi calls them shows) include an array of music, comedy and celebrity interviews.

RadioMoi was the first webcaster to sign an agreement with the RIAA and to be licensed under the Digital Millennium Copyright Act. This license allows RadioMoi to stream copyrighted sound recordings and requires them to make royalty payments. RadioMoi also provides links to artist and record label Web sites and lets listeners purchase albums on the spot.

Radio Moi uses an encrypted form of MP3. To listen to audio, you must use Winplay (the free RadioMoi player). If you have other MP3 players, such as Winamp, installed you may need to change the application associated with the .M3U file type. Otherwise, your MP3 player may attempt to play the

RadioMoi channel—which it can't because of the encryption. (See Chapter 9, *Organizing and Playing Music*, for more information on file type associations.)

Rolling Stone Radio

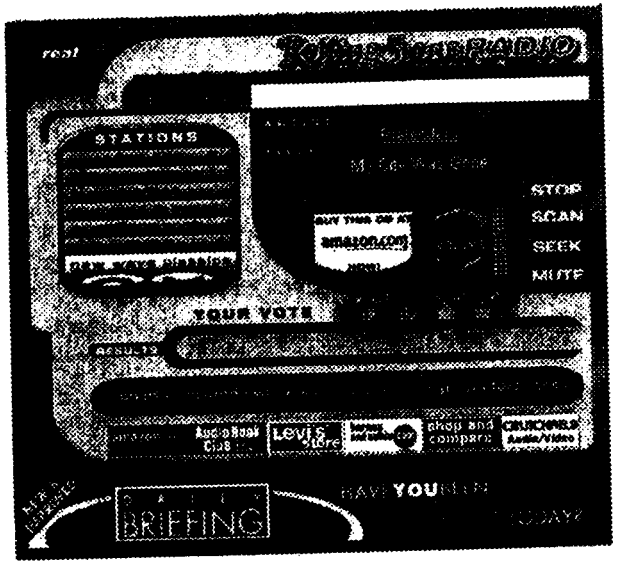
Rolling Stone Radio (*www.rsradio.com*) is Rolling Stone Magazine's Internet radio site. It offers a diverse selection of music channels, including DBRN, which is the David Bowie Radio Network, offering the rock star's favorite music.

Rolling Stone Radio's channels are located on the left side of the player. Only some of them show, so you'll need to use the up and down arrow keys on the player to scroll through them. Once you find a channel you want to listen to, click the **Play** bution. After several seconds, the song will begin to play, and the title and artist name will be displayed. Click on the artist name to see more information about that artist.

You can rate any song while it's playing by clicking on one of the checkboxes labeled 1-5. If you like the music, you can click on a link that will take you to Amazon.com, where you can purchase the album. You can also click on a button to submit a song request to the Rolling Stone Radio DJ.

To play Rolling Stone Radio, you need the RealPlayer G2 and the Rolling Stone Radio tuner, both of which can be downloaded at the rsradio.com site for free.

Rolling Stone Radio Tuner



Spinner

Spinner (*www.spinner.com*) is owned by America Online and offers access to more than 150,000 songs across 100+ music channels, grouped by genre, with programmable presets. You can rate any song that's played, access artist information, and, if you want, purchase the CD. Currently, Spinner doesn't allow you to set up your own station.

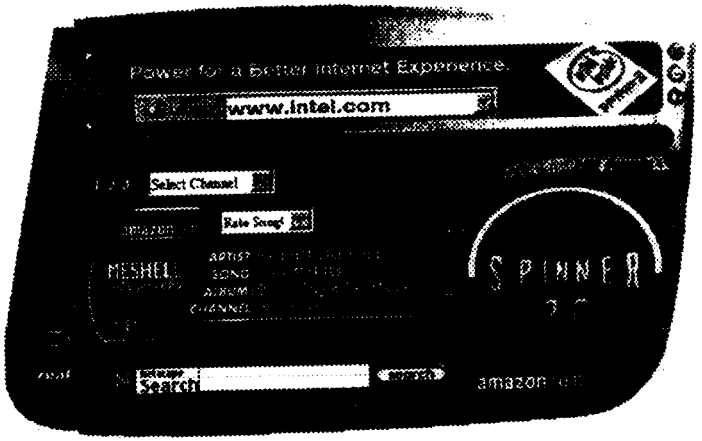
RN-00209

To listen to Spinner on a Windows system, you need to install their stand-alone player. If you're on a Mac or Unix system, Spinner offers a player that runs in conjunction with your Web browser. You'll also need to install the *RealPlayer*.

Streaming MP3

Streaming MP3 has rapidly become the choice for amateur webcasters worldwide. Now, anyone with a PC and an Internet connection can inexpensively stream music to listeners throughout the world, using SHOUTcast or Icecast streaming MP3 software.

Spinner Plus Tuner



SHOUTcast

Nullsoft's SHOUTcast (*www.shoutcast.com*) provides users with a simple way to stream MP3 music to listeners all over the world. SHOUTcast servers can submit their description and status back to the main SHOUTcast server directory, which allows listeners to locate SHOUTcast servers without knowing their IP addresses.

Icecast

Icecast (*www.icecast.org*) is an open source streaming MP3 server, similar to SHOUTcast. It is available for free, including the source code. Because it is open source, useful modifications and additions by users are incorporated back into the main code for the benefit of all users.

MP3Spy

MP3Spy (*www.mp3spy.com*) helps you find SHOUTcast servers and listen to webcasters from all over the world. MP3Spy lists the available SHOUTcast servers and identifies them by music genre and type of programming. When you choose a server, MP3Spy connects you to the server's audio stream and the Web page of the station. When you connect to a SHOUTcast server, you can also chat with the DJ or other users who are listening to the same music. You'll need an MP3 player, such as Winamp or Sonique, to use MP3Spy.

Webcasting Licensing

Internet radio stations can give listeners a high degree of control over the music they hear, but the music industry seems to fear that giving listeners too much control will reduce music sales. Their reasoning seems to be that if people could choose to listen to any song at any time, there would be little incentive for anyone to actually purchase music.

While the recording industry was slow to recognize the potential of downloadable music, it was quicker to recognize the potential (and threat) of Internet radio and lobbied to have laws enacted to protect its interests. The Digital Millennium Copyright Act (sponsored by the recording industry) addresses the issue of webcasting by providing statutory (mandated by law) licenses for webcasters who meet certain conditions. (See Chapter 5, *Digital Music and Copyright Law*, for the requirements for statutory webcasting licenses.)

Some Internet radio sites exist that webcast music illegally, but many webcasters want to be "legal" and are obtaining or have obtained the licensing required. Amateur webcasters are popularizing streaming audio, just like grass roots support and the Internet popularized MP3. But the recording industry is bent on ensuring that proper royalties are paid whenever copyrighted music is played and that music streamed over the Internet doesn't cut in to music sales.

In addition to licensing fees, webcasters are subject to several significant restrictions. For example, while Internet radio listeners can select the songs they want to hear, it is illegal for webcasters to allow them to select a particular song to play instantly, unless the song has been specifically authorized for interactive distribution. Even though listeners can create personalized stations, the site's DJ must rotate the playlists and determine when each song is played.

Webcasters are concerned that these types of restrictions will inhibit their ability to play the music that listeners want to hear, and make it financially unfeasible to operate a radio site. Internet radio is evolving rapidly, and more legislation may be required as it matures. Eventually, more standards and laws will be established and Internet radio will become a major component of our media, just like broadcast radio and television. Until then, it's still a bit like the Wild Wild West.

Item -8-

Proceedings of the 1999 International Computer Music Conference

A Content-Aware Sound Browser

Douglas Keislar, Thom Blum, James Wheaton, and Erling Wold
Muscle Fish, LLC
2550 Ninth St., Suite 207 B, Berkeley, CA 94710, USA
1-510-486-0141 inquiries@musclefish.com
http://www.musclefish.com

ABSTRACT: The SoundFisher™ browser is a cross-platform, single- or multi-user sound-effects database application. It incorporates an audio analysis engine that permits retrieval of sounds based on their acoustical similarity, as well as on traditional keywords and file attributes. Databases can be constructed from sounds on a local filesystem and/or the World Wide Web.

1. Introduction

When studios amass collections of sound effects and other sound files totaling hundreds of gigabytes, intuitive means of organizing and retrieving sounds become imperative. Traditional approaches have required time-consuming manual classification and organization. Describing the sound with keywords, while important, is not enough. What is needed is a system that can automatically compare, classify, and retrieve sounds. A number of researchers have investigated the problem of how to automatically classify and retrieve non-speech audio. (See references.)

This paper describes an audio engine and an end-user application, both already implemented, that permit retrieval of sounds based not only on traditional methods (keywords, soundfile header information, creation date, etc.), but also on the sounds' content—i.e, acoustic attributes. The end-user application is a multiplatform, multiuser sound-effects browser, called SoundFisher. (This paper does not mention the many features of the engine that are unused by SoundFisher. See Wold et al. 1999 for more information.)

2. Audio feature analysis and comparison

This section summarizes our technique for analyzing audio signals in a way that facilitates audio classification and search. For each frame of audio data (25 ms long, with a hop size of 10 ms) we measure a number of acoustic features of each sound. The analysis produces, over the course of the entire sound, a time series where each element is a vector of floating-point numbers representing the instantaneous values of the features. This sort of analysis works best when the sound is homogeneous in character, e.g., a door slam or rain. When analyzing a longer heterogeneous recording, e.g., a news broadcast, one can automatically segment the recording and compute a feature vector for each segment.

2. 1. Frame-Level Features

The following features are currently extracted from each frame: loudness, pitch, brightness, bandwidth, and mel-filtered cepstral coefficients (MFCCs). The first three features were discussed in Keislar et al. (1995). Bandwidth is computed as the magnitude-weighted average of the differences between the spectral components and the centroid. A vector of MFCCs is computed by applying a mel-spaced set of triangular filters to the STFT and following this with the discrete cosine transform.

Since the dynamic behavior of a sound is important, the low-level analyzer also computes the instantaneous derivative (time differences) for all the aforementioned features.

2. 2 Higher-Level Features

From the time series of frame values, we extract higher-level information. We compute the mean and standard deviation of the frame-level time series for each parameter, including the parameter derivatives. When computing the mean and standard deviation, the frame-level features are weighted by the instantaneous loudness so that the perceptually important sections of the sound are emphasized.

The user can present the system with a single sound for comparison, or with examples of a class of sound. In the latter case, we can infer something from the variability of the parameters across the different recordings. For example, there may be several samples of oboe tones, each at a different pitch. If one of these is presented as an example of an oboe sound to the system, the system has no *a priori* way of determining that it is the timbre of the sound that determines the class, rather than the particular pitch of this sample. However, if all the samples are presented to the system, the variability of the pitch can be noted across the samples and then used to weight the different parameters in comparison. This information can then be used when comparing new sounds to this class. This

variability is stored in the standard deviation portion of the class's feature vector.

For this single-Gaussian statistical model, the distance measure used is essentially the Euclidean distance between the two sounds' feature vectors, with each dimension scaled by its standard deviation. The user is given the ability to apply additional weights to the different features, which is useful when certain features are known to be more (or less) pertinent to the task at hand.

3. The SoundFisher Sound-Effects Browser

This section presents the SoundFisher audio browser, which runs on the Macintosh, Windows (95, 98, and NT), and UNIX (Solaris, SGI) platforms. Written in Java, the SoundFisher GUI communicates with an included audio-analysis and database engine written in C.

Figure 1 shows the GUI for the application after a search has been performed. The row of buttons across the top provides functions such as "forward" and "back," allowing navigation to previously displayed query results. Below the buttons is the query area, and the bottom portion of the window displays the query results.

A database is built up by adding URLs to it (either local files or Web addresses.) Directories can be added recursively; in a single step one can add all the sound files on a given disk, for example. The supported audio file formats include WAV, AIFF, AU, and Sound Designer II. When sounds are added, the engine analyzes the audio in the file or URL and stores the resulting feature vector in the database. Long sound files can be automatically segmented. In addition, "thumbnails" of sounds can optionally be generated. A thumbnail is a low-resolution, optionally truncated version of the source sound file. Thumbnails are useful for auditioning search results when the original sounds are offline.

The data record for each sound includes not only the acoustic feature vector but also soundfile information (sample rate, format, number of channels, duration, etc.), date, and textual keyword and comment fields. The text fields can be applied recursively when adding a directory. In addition, the user can define and add new text fields. Text fields can be edited at any time.

Users can create hierarchical categories of sounds. The default categories mimic the filesystem organization of the added directories, but the category names and hierarchy can be easily edited using a familiar graphical paradigm (e.g.,Windows Explorer or the Macintosh Finder's List view). These categories are arbitrarily defined by the user. The user can also create "classes": sets of sounds whose acoustical feature vectors are close to sounds that the user has provided as a training set.

Below the top row of buttons is the query area, which is reminiscent of the Find File utilities on Mac and Windows. Multiple criteria can be combined with a Boolean AND operation. A query is formed using a combination of constraints on the various fields in the database schema as well as "query by example" (comparison to a selected sound). For example, Figure 1 illustrates a query based on similarity to a selected sound (the noise of a crowd), in combination with a constraint based on a data field (duration). As indicated, the search can operate over the entire database, or it can apply to the currently displayed or currently selected records.

The bottom portion of the window displays the current records (often, the result of a query). Sounds can be auditioned by double-clicking, and multiple selections are possible. These results can be viewed in one of three ways: as a list (Figure 1), hierarchically by category, or as a 2-D plot (Figure 2). In the 2-D plot, the axes can be various acoustic attributes or the begin and end times of the sounds. The begin and end times are useful for automatically segmented sound files; by choosing begin time for an axis, one can view the temporal trajectory of a particular acoustic feature.

For more information, see the Muscle Fish Web site, www.musclefish.com.

References

Dubnov, S., N. Tishby, and D. Cohen. 1996. "Clustering of Musical Sounds using Polyspectral Distance Measures." *Proceedings of the International Computer Music Conference 1995*, pp. 460-463.

Feiten, B. and S. Günzel. 1994. "Automatic Indexing of a Sound Database Using Self-organizing Neural Nets." *Computer Music Journal* 18(3): 53-65.

Foote, J. 1997. "A Similarity Measure for Automatic Audio Classification." *Proceedings AAAI 1997 Spring Symposium on Intelligent Integration and Use of Text, Image, Video and Audio Corpora*, Stanford, California.

Hashimoto, S., H. Qi, D. Chang. 1996. "Sound Database Retrieved by Sound." *Proceedings of the 1996 International Computer Music Conference*, pp. 121-125.

Keislar, D. ,T. Blum, J. Wheaton, and E. Wold. 1995. "Audio Databases with Content-Based Retrieval." *Proceedings of the 1995*

mputer Music Conference, pp.

ler, and W. Effelsberg. 1996.
o cor analysis," Tech. Rep.
versit, . Mannheim.
Slaney. 1997. "Construction
f a Robust Multifeature
scriminator." *Proceedings*
nich, Germany.
D. Keislar, and J. Wheaton.

1996. "Content-Based Classification, Search, and Retrieval of Audio." *IEEE Multimedia* 3(3): 27-36.

Wold, E., T. Blum, D. Keislar, and J. Wheaton. 1999. "Classification, Search, and Retrieval of Audio." In Furht, B., ed. *Handbook of Multimedia Computing*, pp. 207-227. Boca Raton, Florida: CRC Press.

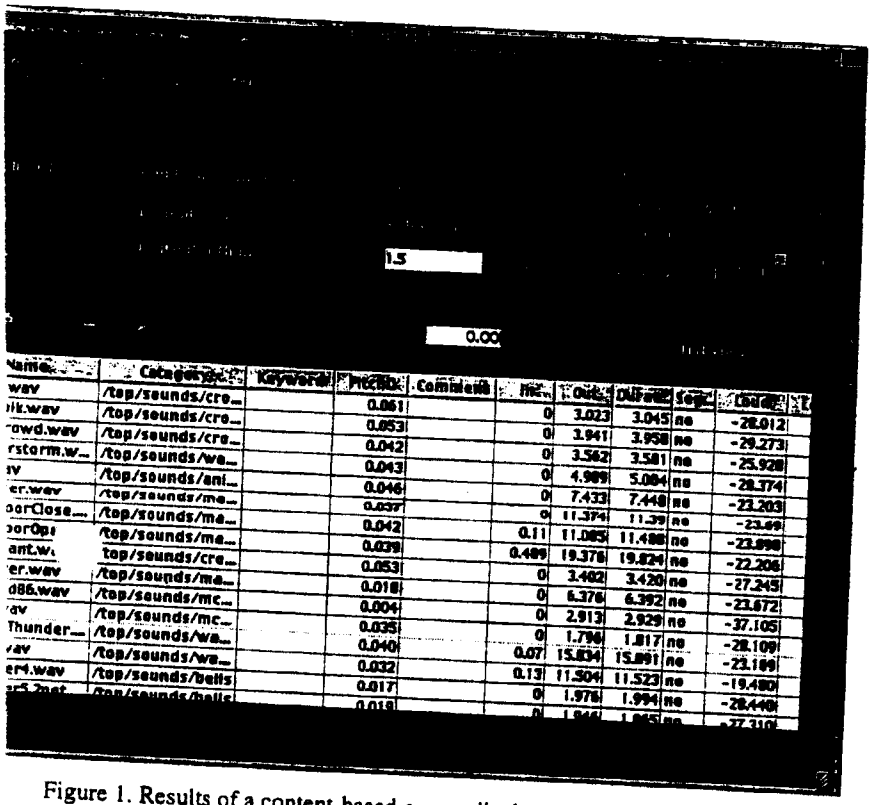


Figure 1. Results of a content-based query, displayed as a list.

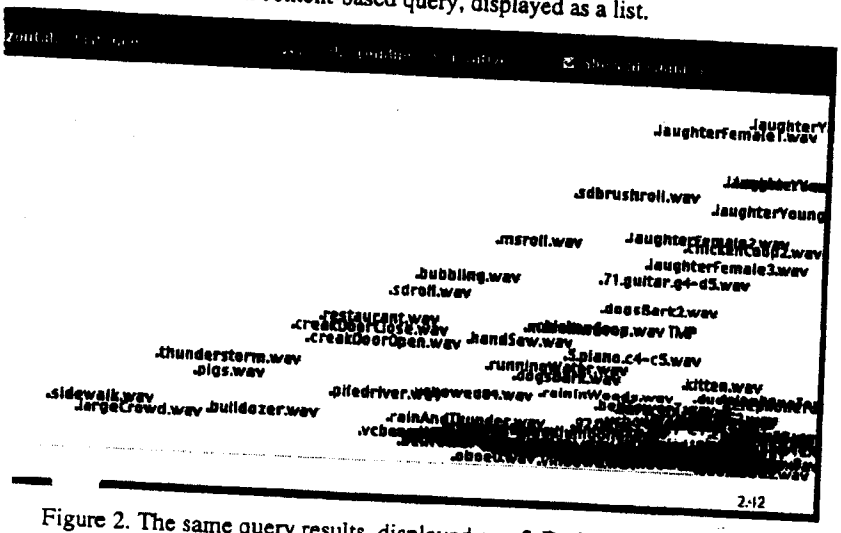


Figure 2. The same query results, displayed as a 2-D plot.

Workshop Program

Fourteenth International Joint Conference
on Artificial Intelligence

Intelligent Multimedia Information Retrieval

Mark Maybury, Chair

WORKING NOTES

SATURDAY, AUGUST 19, 1995

Montréal, Québec

Audio Databases with Content-Based Retrieval

Thom Blum,
Douglas Keislar,
James Wheaton,
and Erling Wold

Muscle Fish
2550 Ninth Street, Suite 207 B
Berkeley, CA 94710, USA

e-mail: inquiries@musclefish.com

Abstract

Despite vast research and development efforts in such diverse domains as digital signal processing, psychoacoustics, speech recognition, computer music, and multimedia databases, there is a paucity of literature that addresses the issue of automatic classification of sounds (whether real-world sounds or synthetic sound effects). In this paper we suggest that much of the necessary knowledge exists today but needs to be redirected or refocused to yield the solutions to these specific problems. After surveying some related research, this paper presents our work-in-progress, developing an analysis engine and a client application that would help automate the process of sound classification, retrieval, and selection.

.. Introduction

Anyone who has ever heard a sound and tried to describe it knows of the difficulties. Words are woefully inadequate to convey the essence of a sound. It has been said, for example, that writing about music is like dancing about architecture. This being the case, people often resort to describing sound by the use of simile ("it sounds like a herd of elephants") or through the use of onomatopoeia (a film producer working at the side of a sound designer might ask for "a good *thwack* in that face-hit."). At other times people will describe a sound by referring to some emotion that it evokes ("it has a mournful sound"). A musician might describe a sound using musical terms like "crescendo from mezzo-piano to fortissimo." Similarly, an acoustician might refer to a sound's acoustical attributes ("it has an exponential decay and the energy is concentrated in the upper partials"). And of course, sometimes descriptive adjectives are evocative ("a shimmery sound"). When words fail to describe a sound, the sound might best be described by comparing it to another ("this sound is very similar to that sound," or "find me another sound that sounds like this one").

But who really cares about finding a sound, whether by describing it verbally or by comparing it to another sound? Ask the film sound-effects designer who is struggling to finish off that large-restaurant ambient sound for the current scene while the producer runs out of patience. Or ask those computer animators that wish they had a system that would automatically find and mix some appropriate sounds to accompany their latest short film. And ask the technology-savvy sales representative who has finally been convinced that multimedia can breathe life into the presentation (due tomorrow) and who is now trying to find a sound with punch to conclude the slide show.

Such users are greatly hampered by existing sound-effects databases (often referred to as "librarians"). These databases typically permit the user to associate a limited number of textual keywords and/or descriptions with each sound. A sound can often be placed into a category, but generally cannot appear in more than one category. This is a severe limitation, since the classification of any sound will often change depending upon the way that sound functions within a specific category (for example, within a given document or composition).

While words can sometimes serve as sufficient keys for the retrieval of data, there are many cases where one would like to query the source more directly. *Content-addressable databases* or *content-based retrieval* are the

terms usually used to describe this kind of information storage and retrieval system. Such systems permit searches for features, keys, or triggers extracted directly from the data (as opposed to being one step removed from the data, as in a keyword-only description of that data). While this capability has ceased to be a problem for text-only databases, the race for a solution to this problem in the area of multimedia document management is just starting. As usual, sound has taken a back seat to image (both moving and still), but it will not be ignored for long. As more and more sounds become available to creators of multimedia works, there is an increasing need to quickly and efficiently locate a particular sound or a set of similar sounds. Giving authors of multimedia documents a choice of hundreds or thousands of sounds without the tools to find them is a little like casting someone adrift in a rowboat without any oars.

In this paper, we will introduce some novel approaches to analyzing, cataloguing, and retrieving sounds from an audio database. Although the approaches we will present are somewhat new, the techniques of time-varying signal analysis and processing on which our approach is based have been in existence (or evolving) for years. One of the main goals of this paper is to suggest ways in which sounds may be retrieved from repositories by using any one or a combination of objective (acoustic) metrics, by specifying subjective perceptual features, or even by selecting or entering a reference sound and asking the database to retrieve all sounds that are similar (or dissimilar) to it. The four main sections of this paper review the related literature, explain the signal-analysis techniques that we propose for a sound database, present the database schema, and "walk through" the functions of our suggested database browser.

2. Previous Research

Sound Taxonomies

During the last four decades, numerous attempts have been made to develop taxonomies of sounds, including musical and environmental sounds [Schaeffer 1966, Schafer 1980, Tenney 1961, Vertegaal and Bonis 1994]. This interest of musicians and psychologists has shed some light on the analysis and classification of sound, be it by objective or subjective measures.

Timbre Analysis

Sounds are traditionally described by their pitch, loudness, duration, and timbre. The first three of these perceptual attributes are well-understood and fairly easily measured. Timbre, on the other hand, is an ill-defined attribute that encompasses all the distinctive qualities of a sound other than its pitch, loudness, and duration. The effort to discover the components of timbre underlies much of the previous psychoacoustic research that is relevant to content-based audio retrieval [Helmholtz 1885, Risset and Mathews 1969, Plomp 1976, Grey 1977, Gordon and Grey 1978, Wessel 1979].

Salient components of timbre include the amplitude envelope, harmonicity, and spectral envelope. The attack portions of a tone are often essential for identifying the timbre. Timbres with similar spectral energy distributions (as measured by the centroid of the spectrum) tend to be judged as perceptually similar. However, research has shown that the time-varying spectrum of a single musical instrument tone cannot generally be treated as a "fingerprint" identifying the instrument, because there is too much variation across the instrument's range of pitches, and across its range of dynamic levels.

Source Separation

Simultaneous sound sources present a huge obstacle for any sound-analysis environment. Approaches to separating simultaneous sounds typically involve either Gestalt psychology [McAdams 1984, Bregman 1993, McAdams 1993] or non-perceptual signal-processing techniques [Moorer 1975, Wang 1994]. For musical applications, automatically parsing a monophonic melody is feasible, but a completely general-purpose, polyphonic pitch-tracking algorithm might well be an intractable problem, recent efforts notwithstanding [Moorer 1975, Chafe *et al* 1985, Depalle *et al* 1993].

Sound Librarians and Editors

As mentioned earlier, most existing sound databases and sound librarians suffer from a retrieval paradigm that is limited to keywords or mutually exclusive categories, with no possibility of content-based retrieval. Also, data entry in existing systems is intensive: there is no automatic analysis or classification of sounds.

Various researchers have discussed or prototyped graphical sound editors capable of extracting musical structure from a sound [Buxton *et al* 1981, Chafe *et al* 1982, Foster *et al* 1982]. The goal was to allow queries such as "find the first occurrence of the note G-sharp." Unfortunately, most of this research came to a halt after the introduction of MIDI (Musical Instrument Digital Interface), which eliminates the need for this functionality in the case where the sound is generated by a MIDI instrument.

The Intuitive Sound Editing Environment (ISEE) is a recent software package for controlling MIDI synthesizers [Vertegaal and Bonis 1994]. Its user interface is based on the notion of timbre space [Wessel 1979]—that is, continuous control of a small number of orthogonal, device-independent timbral parameters. The parameters are "overtones" (harmonicity), "brightness" (spectral energy distribution), "articulation" (control of spectral transients and persistent noise), and "envelope" (the speed of the amplitude envelope). These parameters could also be applied to the analysis of digital audio, an approach that would bear some similarity to the analysis techniques presented in this paper.

3. An Analysis Engine for Content-Based Retrieval of Audio

In this section of the paper, we present a general paradigm and specific techniques for analyzing audio signals in a way that facilitates content-based retrieval. (Section 5 of the paper describes a client application with a graphical user interface for retrieving the audio data.)

3.1 Introduction To The Analysis Technique

By content-based retrieval of audio, one can mean a variety of things. At the simplest level of implementation—but the least simple level of usage—one could retrieve a sound by specifying the exact numbers in an excerpt of the sound's sampled data. At the next higher level of abstraction, the retrieval would match any sound containing the given excerpt, regardless of the data's sample rate, quantization, compression, etc. At the next level, the query might involve frequency-domain information or other acoustic attributes that can be directly measured. Finally—at the most difficult level of implementation but potentially the most user-friendly level—the query could include perceptual (subjective) properties of the sound.

It is this final level—perceptual properties—with which we are most concerned. (The implementation of the first two levels is conceptually straightforward and need not be discussed here.) Some of the aural (perceptual) properties of a sound, such as pitch, loudness, and brightness, correspond closely to measurable attributes of the audio signal, making it logical to provide fields for these properties in the audio database record. However, other aural properties (for instance, "scratchiness") are more indirectly related to easily measured acoustical attributes of the sound. Some of these properties may even have different meanings for different users. (The phenomenon of synaesthesia is an extreme case of subjectivity: a user might call certain sounds "blue" and others "red."). To support subjective properties, the database record format should be user-extensible.

To be able to use different perceptual criteria to retrieve a sound, we first measure a variety of acoustical attributes of each sound. This set of N attributes is represented as an *N*-vector. In text databases, the resolution of queries typically requires matching and comparing strings. In an audio database, we would like to match and compare the sort of aural properties described above (such as "scratchiness"). For example, we would like to ask for all the sounds similar to a given sound or that have more or less of a given property. To guarantee that this is possible, the space of *N*-vectors should satisfy the following constraints for each aural property to be used in retrieval:

1. Sounds which differ in the aural property should map to different regions of the N-space. If this were not satisfied, the database could not distinguish between sounds with different values for this property.

2. If the user ranks a set of sounds in increasing amounts of the given aural property, these sounds should map approximately to a smooth path in some M-dimensional projection of the N-space where $M \leq N$. Since we use a linear model, we have the additional restriction that the smooth path should be approximately linear. Note: In the case where the aural property is binary—that is, where a sound either has the property or it doesn't—the linear constraint is not so important.

Since we cannot know the complete list of aural properties that users may wish to specify, it is impossible to guarantee that our choice of acoustical attributes will meet these constraints. However, we can make sure that we can meet these constraints for many useful aural properties.

3.2 Acoustical Attributes

The following aspects of sound are analyzed:

• *Pitch.* Pitch is estimated by taking a series of short-time Fourier spectra. For each of these frames, the frequencies and amplitudes of the peaks are measured and an approximate weighted greatest common divisor algorithm is used to calculate the pitch (expressed as log frequency). The pitch algorithm also returns a pitch confidence value which can be used as a measure of "how pitched" the sound is.

• *Harmonicity.* This parameter distinguishes between harmonic spectra (e.g., vowels and most musical sounds), inharmonic spectra (e.g., metallic sounds), and noise (spectra that vary randomly in frequency and time).

• *Loudness.* Loudness is approximated by the signal's RMS level in decibels, which is calculated by taking a series of windowed frames of the sound and computing the square root of the sum of the squares of the windowed sample values. (This method does not account for the frequency response of the human ear; if desired, the necessary equalization can be added by applying the Fletcher-Munson equal-loudness contours.)

• *Brightness.* Brightness is computed as the centroid of the short-time Fourier spectra.

• *Formants.* Formants are computed by smoothing the short-time Fourier spectra and looking for broad peaks. (We use the word "formants" loosely to refer to broad spectral peaks, which may or may not be constant as the fundamental frequency varies.) The formants are parameterized by logarithmic values of frequency, magnitude, and width.

All the above aspects of sound can vary over time. The trajectory in time is computed during analysis but not stored as such in the database. However, for each of these trajectories, several parameters *are* computed and stored, including:

• Average.

• Maximum and minimum.

• Variance.

• Autocorrelation. This is a measure of the smoothness of the trajectory. This can distinguish between a pitch glissando and a wildly varying pitch (for example), which a simple variance measure cannot.

• Parameters relating to the shape of the smoothed trajectory: critical points, number of inflections, attack and decay time (of loudness trajectory).

In addition, the duration of the sound is stored. The *N*-vector of measured attributes thus consists of duration plus the parameters just mentioned (average, maximum, minimum, autocorrelation, shape parameters) for each of the aspects of sound given above (pitch, harmonicity, loudness, brightness, and formants). If some of these parameters are found to be less useful for certain of the aspects of sound, those attributes can be omitted, for efficiency.

3.3 Training The System and Retrieving Data

Continuous Properties

As we mentioned above, some aural properties will directly relate to the measured attributes above. However, we need a method to teach the system about new aural properties, especially those that are subjective and can vary between different users.

To train the system, the user picks a set of sounds which show varying amounts of the property in question. The user sorts the sounds according to their perceived ranking, assigns an approximate numerical value of the property for each sound, and submits them to the system. When training by example, the more examples the user has, the better the system's understanding will be. It would also be best if the user submitted examples which covered a wide range of values for the property.

For each sound $s[j]$, $j=0$ to $M-1$, the system computes the N-vector a, if it is not already computed. (M is the number of sounds to be analyzed, and N the number of acoustical parameters.) To find the relationship between the aural property $p[j]$ of each sound and the measured attributes $a[j]$, we use a standard linear regression model with parameter vector b. That is:

$$p[j] = b^T a[j] + e[j]$$

where e is the error in the model. (The superscript T indicates a transposed matrix.)

Given $p[j]$ and $a[j]$, the b parameters can be estimated using least squares, which is the unbiased, minimum variance estimate for b. Note that the elements of b can be reported to the user to indicate which elements of a are most significant to the aural property under consideration.

The algorithm computes the variance of e, which can be examined to give the user an indication of how well the model fit the data. At some threshold, the system could indicate that the training failed—that is, the user needs to supply more training sounds, or the currently measured attributes do not meet the constraints (listed in Section 3.1, above) for the specified aural property. In the latter case, the property might be an uncomputable property (e.g., "how much I like the sound"), or there might be further measurements which could be added to the repertoire to increase the usefulness of the database.

Once the mapping between measured attributes and the aural property is understood by the system, the mapping is applied to each sound in the system (or each of the sounds currently of interest to the user). The name and value of the property is included in the database record for that sound. Once this is accomplished, it is straightforward to select sounds from the database with queries relating to the property. For example:

- *Query by value.* Retrieve all the sounds with values of property p_0 greater than 0.9 and property p_1 less than 0.2.

- *Query by example:* Retrieve all the sounds similar to this sound with respect to property p_0. "Similar to" means "within some delta of." Retrieve all the sounds with less p_1 than this sound.

• *Organization/Browsing:* Sort the current sounds by property $p0$. Group the current sounds by properties p_0 and p_1.

Binary or Discrete Properties

In these cases, the property is used as a classifier. In the binary case, a sound either has this property or does not, and in the more general discrete case, the sound falls into one of several categories. To train the system, the user selects examples of sounds which have the property or do not (in the binary case) or which illustrate the different categories (in the general discrete case).

For each sound, the a vectors are computed if they have not already been computed. The mean vector μ and the covariance matrix R for the a vectors in each category are then calculated. The mean and covariance are given by:

$$m = (1/M) \sum a[j]$$
$$R = (1/M) \sum (a[j]-\mu)(a[j]-\mu)^T$$

where M is the number of sounds in the summation.

When a new sound needs to be classified, a likelihood value is calculated using the multivariate normal distribution:

$$exp((-1/2)(a-\mu)^T R^{-1}(a-\mu))$$

To speed up the process, a simpler likelihood value could be used—namely, just the argument of the above exponential, with appropriate normalization.

If the likelihood for the category is above a threshold, the sound is determined to be in that category. If there are several mutually exclusive categories, the sound is placed in the category with the highest likelihood.

3.4 Miscellaneous issues in sound analysis

Segmentation

The above discussion handles the case where each sound is a single gestalt. Longer recordings need to be segmented before using the retrieval features above. Segmentation is accomplished by applying the acoustic analyses above to the signal and looking for transitions (sudden changes in the measured attributes). The transitions define segments of the signal, which can then be treated like individual sounds.

For example, if the user of the system trained it using applause sounds, a recording of a concert or other performance could then be automatically scanned for applause sounds to determine boundaries. Similarly, after training the system to recognize a certain speaker (primarily by the formant structure), a recording could be segmented and scanned for all the sections where that speaker was talking.

Sonification

Non-audio data can be converted into audio data using a mapping between arbitrary data parameters and audio parameters. This is known as sonification (the sound analogue of visualization). Once data is put into a meaningful audio form, it can be treated exactly like audio data, and one can make use of the audio retrieval methods above. For example, a scientist or a medical technologist might find it easier to identify the "sound" of a certain sonified function than to describe its numerical or visual characteristics.

Neural Nets

Neural nets are another possible way to find the mapping between between acoustical attributes a and perceptual properties p. The advantage of neural nets is that they can discern non-linear mappings. The disadvantage is that it is difficult to see what is going on "inside" the net. Thus, one cannot estimate how good a model the system has discovered (how well it understands), and one cannot see which measured attributes correlate most strongly with p.

User control of analysis

In some cases, the user might know which measured attributes are relevant to the aural property in question. The analysis of an aural property can be made faster and more reliable in these cases by allowing the user to constrain the analysis to a subset of all the measured attributes.

One can also imagine making the analysis engine extensible, by allowing a user to "plug in" new analysis algorithms. This feature would complicate the software design and implementation.

4 The Database Schema

This section describes a data record structure suitable for use with the analysis engine discussed above.

Each entry in the database points to and describes a particular sound. Fields in each entry include attributes such as the sound's name, the list of categories and subcategories that the sound is a member of, the list of features (as computed by system analysis routines), a keyword list (user-defined), and links to related files and documents which exist outside the database proper. The database schema is summarized in Figure 1.

The "name" field contains a string that identifies the sound. If no name is supplied, the name of the sound file is used.

"Categories" are arbitrary optional classifiers whose existence is unknown to the audio analysis engine. They can be used to identify the project—that is, the group of users or the works (such as compositions or soundtracks) that make use of the sound. Or they can represent other user-specified classification schemes—such as a taxonomy of musical instruments. A sound may be a member of many (or no) categories. For example, a single saxophone sound might be be contained in the categories "marketing," "development," "woodwinds," and "jazz." Categories can recursively contain subcategories; a character such as "/" serves as a delimiter for subcategory names. A subcategory might refer to a team of users within a department, or to some subdivision of the work, such as one scene in a film or one track of a soundtrack. (Recursive hierarchies could conceivably be useful for aural properties as well, but for simplicity we allow only categories to be specified hierarchically.)

"Keywords" are optional user-supplied words, usually descriptive adjectives or nouns, that help tag a sound for queries. The only essential difference from categories is that keywords cannot be hierarchical.

The "features" field of the database schema contains the results of the acoustical analyses described earlier. In other words, this field stores the "measured attributes" vector. (As was discussed in Section 3.2, the measured attributes include the sound's duration, as well as the average, range, variance, autocorrelation, and shape parameters for each of the sound aspects: pitch, harmonicity, loudness, brightness, and formants.) This field also contains the user-specified perceptual descriptions (the "aural properties" of Section 3), which can be thought of as subjective overlays on top of the objective measured attributes. Note that the user cannot directly edit the values of either the measured attributes or the aural properties; the system is responsible for maintaining these values. Later, we will describe how the feature list is used by the browser, and how users can define new aural properties.

The aural properties field just contains the names of the aural properties. Aural properties are fully specificied by a separate schema (not illustrated), which includes the name of the property, the name of the user who defined the

property (by training the system to understand it), the mathematical mapping of the measured attribute vector to the aural property, the date the property was last defined or modified, and links to the sound files (and/or the sound names) that were referenced in defining the property.

In many applications, sound files can have heterogeneous data formats, such as different sample rates, bits per sample, or compression schemes. Most sound file formats include a header that fully specifies the data format. However, this information could also be stored in the data record. Although such information can be of interest to users, we have omitted it from the illustrations for simplicity.

Also, the database schema presented here doesn't distinguish between the data of different users. A full-blown multi-user system might have fields specifying the user under the aural properties, comments, categories, and keywords fields. This would allow two different users to define two distinct subjective aural properties that happened to have the same name (such as "buzzy"). Users could decide to hide their data from others or share it.

```
name:
        defaults to sound file name

dates:
        created:
        analyzed:
        accessed:

features:
        analyzed by the audio analysis engine
        measured attributes:
            objective attributes of the sound
        aural properties:
            subjective perceptual features

keywords:
        just text, user-supplied, non-hierarchical, optional

categories:
        user-supplied, optional text,
        hierarchical OK (delimiter defaults to "/")

comments:
        system-supplied text
        user-supplied text
        voice annotations
        graphic annotations

history:
        sound file revision history

links:
        sound file
        music score file
        synthesizer "patch"
        source data file for sonification
```

Figure1. Database schema

5 A Tour Of The Database Browser's Functionality

In this section, we present the functionality and user interface of an application that retrieves audio from a database using the analysis engine and database schema described above. This front-end database application lets the user browse through sounds or search for them using queries. In addition, it permits general maintenance of the database's sound entries: adding, deleting, analyzing, and describing sounds. It also lets the user define new "aural properties." The browser described here is merely one of a number of possible applications that could "sit on top of" the database and audio analysis engine.

A series of mock-up diagrams (Figs. 2–6) will serve as a "tour guide" of the application's functionality. These diagrams depict the application as having a menu-based user interface, but in practice such an interface might not always be the most suitable. For example, a sound designer or editor who is accustomed to audio equipment might find a menu-based interface less intuitive than one with physical knobs, buttons, sliders, and perhaps even a piano-like keyboard. For such a user, having to switch interface modes and devices can be highly disruptive. (A separate paper covering these user-interface issues is planned.) Note, however, that even if the browser used a menu-based interface, it might look nothing like the one illustrated here. The purpose of this section is to describe the *core functionality*, not to specify the user interface.

5.1 Outer Window

Figure 2 shows access to all of the browser's functionality: it depicts the application with all menus engaged. Menus on the outer (top-level) window can be used to search for sounds and define new aural properties. The specific procedures supporting these operation are described in subsequent sections. The inner window displays a hierarchy of sorted lists, arranged as successive columns from left to right.

The top-level window's *Query* menu enables the user to search for sounds, querying by value or by example. The *Aural Properties* menu lets the user define new aural properties or modify existing ones. The *Analysis* menu allows the user to view and control the execution of analyses. The *Windows* menu is discussed below. Standard cut/copy/paste functionality is provided by the *Edit* menu.

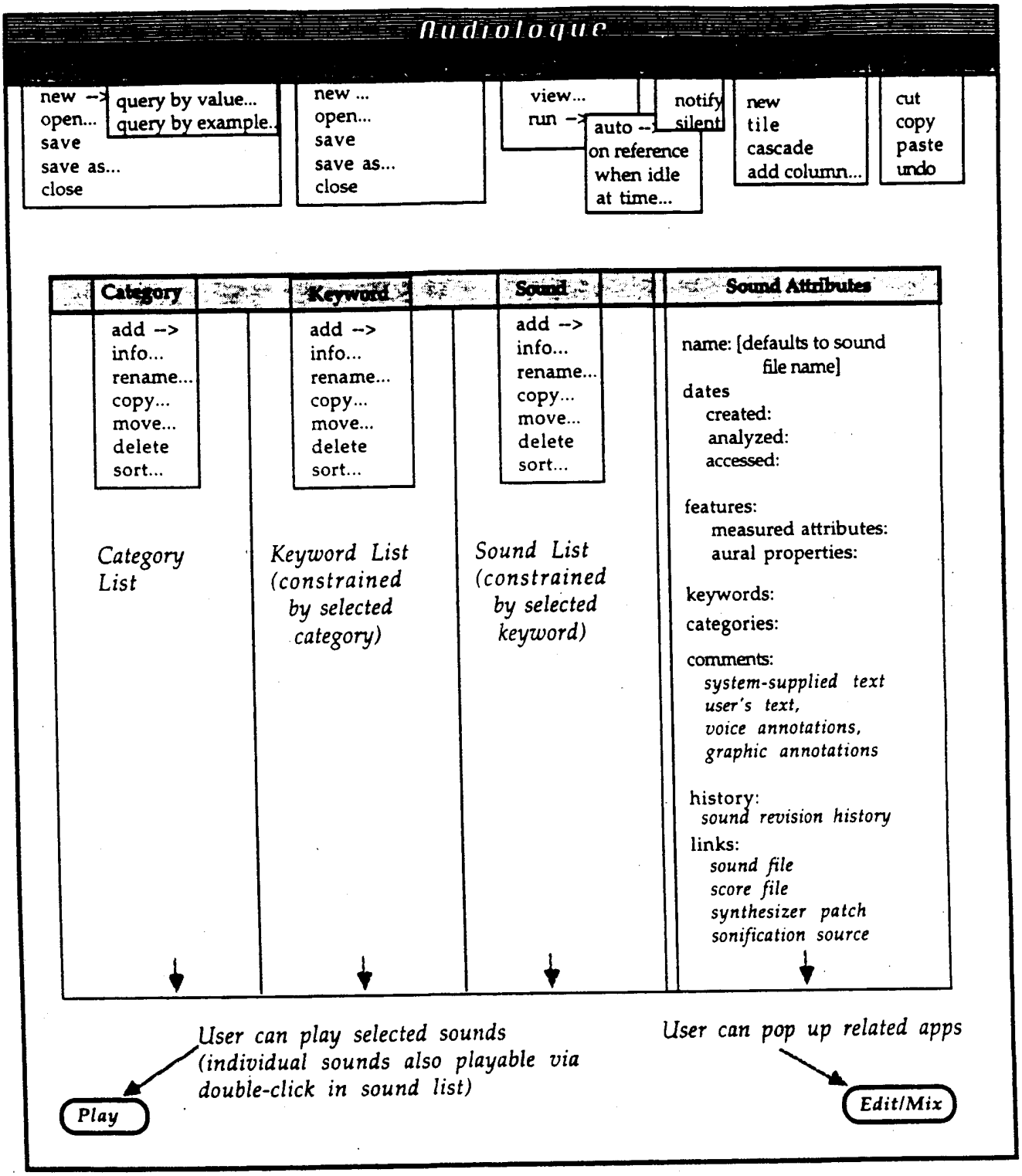


Fig. 2 - An overview of the sound browser. Fully engaged menus show the basic functionality of the browser. Outer (top-level) window can contain multiple views of the database (inner window). Each view consists of multiple scrollable columns that list data field values such as categories and sounds.

5.2 Inner (list) Windows

The displayed ordering of the columns in the hierarchy is under the user's control. (See inner multi-list window, Figure 2.) By default, there are two movable columns—Category and Sound. (The Sound Attributes column is always present and is fixed in the rightmost position.) However, the user can choose what sorts of data fields to organize the browser by. Almost any data field may be assigned to any column, and there may be any number of columns.

For example, the leftmost column might be "Category," the middle column "Keyword," and the rightmost "Sound." These correspond to the "categories," "keywords," and "name" fields of the database schema. The content of the columns is always selection-driven: selecting an item in the leftmost list will result in display updates to the middle list; subsequently, selecting an item in the middle list will update the list displayed on its right, etc. Thus, in this example, the leftmost column would show all the categories in the database; the "Keyword" column would display all the keywords used by sounds in the currently selected category; and the "Sound" column would display the names of all sounds that use the currently selected keyword.

To change the ordering of the columns, the user can simply drag a column to its new location (to the left or right) To add columns, the user can choose the *add column*.. command from the top-level *Windows* menu, which brings up a panel asking the user to specify what data field to display in the column. Cut, copy and paste can be used on columns.

Because categories can be hierarchical, the Category column displays subcategories as indented words. Any depth of subcategories is possible; words that are indented too far to read can be revealed by scrolling the column horizontally (not illustrated). The subcategories can be hidden by a mechanism similar to that used to hide a folder's contents in the Macintosh Finder's "View by Name" mode.

Menus are associated with each list in the multi-list window (Figure 2). These enable standard operations on the selected item(s) in the list. For example, by choosing *add*... in the Category column, the user can create a new category (or subcategory); and by choosing *add*... in the Sound column, a new sound file can be entered into the database. The objects in these columns can be renamed, copied, deleted, and so on. One can also get (and set) annotative information about the objects using these menus.

The column in Figure 2 labeled *Sound Attributes* is a scrolled text window containing the database entry of each selected sound(s). This window is not shown if the multi-list window does not contain a Sounds column. Unlike the other columns, the *Sound Attributes* column cannot be moved relative to the other columns; it is always rightmost and conceptually detached (as indicated by the double line separating it from the column to its left). Some of the fields are editable. In the case of multiply selected sounds, the *Sound Attributes* column contains all the sounds' information, and each sound is accessed by successively clicking the Next and Previous buttons (not illustrated). To apply edits to all the selected sounds (for example, to add the same comment to all of them), you can use the *info*... command in the Sounds column.

It is often useful to have simultaneous displays of the hierarchy, each with a slightly different ordering. Through the *Windows* menu (at the top of the outer window), the system provides methods for generating new list windows. Figure 3 shows two multi-list windows. The top window displays the default list ordering: by category, by sound. The bottom list window organizes the hierarchy by sound, by keyword, by category. The latter ordering is useful when one wants to know all the keywords that have been defined for a particular sound, and all the categories containing sounds tagged with that keyword. Likewise, the user can specify an ordering that displays by sound, by category, by keyword. Such an ordering is useful for browsing all of the categories that contain a particular sound. One could even place two keyword columns side-by-side, to see what other keywords are used in any record along with the selected keyword.

Note that the two multi-list windows in Figure 3 are independent; they need not show the same sound. A sound selected in one window can be dragged and dropped into an appropriate column of the other window. For

example, you could drop a multiple selection from the Sound column of one window into the Category column of another window, in order to add the sounds to that category.

5.3 Play and Edit/Mix Buttons

The button labeled *Play* at the bottom of Figure 3 enables quick previewing of the selected sound(s).

To provide the greatest utility when using the browser, inter-application communication is a desirable feature. The button labeled *Edit/Mix* launches and communicates with an external sound editing application. This editor displays the currently selected sound's waveform and allows the user to cut, copy, and paste portions of it, as well as manipulate it in other ways. (Note that if the user saves an altered sound to the same sound file, the information in the database can become invalid. To address this problem, the sound files can be write-protected, or their modification can trigger a new analysis.)

The editor also functions as a mixer. When *Edit/Mix* is clicked, the browser sends the editor the currently selected sounds, which can be mixed together into a new sound. Because the user can create a new list window in the browser and drag sounds to it from other list windows, it is easy to build up a "mix list"—that is, a list of sounds to be mixed together—before sending them to the mixing application.

This ability to systematically arrange sounds transforms the browser into a *digest*. We refer to this facility of semi-automatic assembly of mixed sounds as "grazing." While grazing starts from the same passive (read-only) behavior as browsing, it leads the application towards a more active (generative) role. To refine the mix, the user may wish to edit the mix-list by directly interacting with the editor application. Grazing is intended as a tool for quickly assembling test mixes.

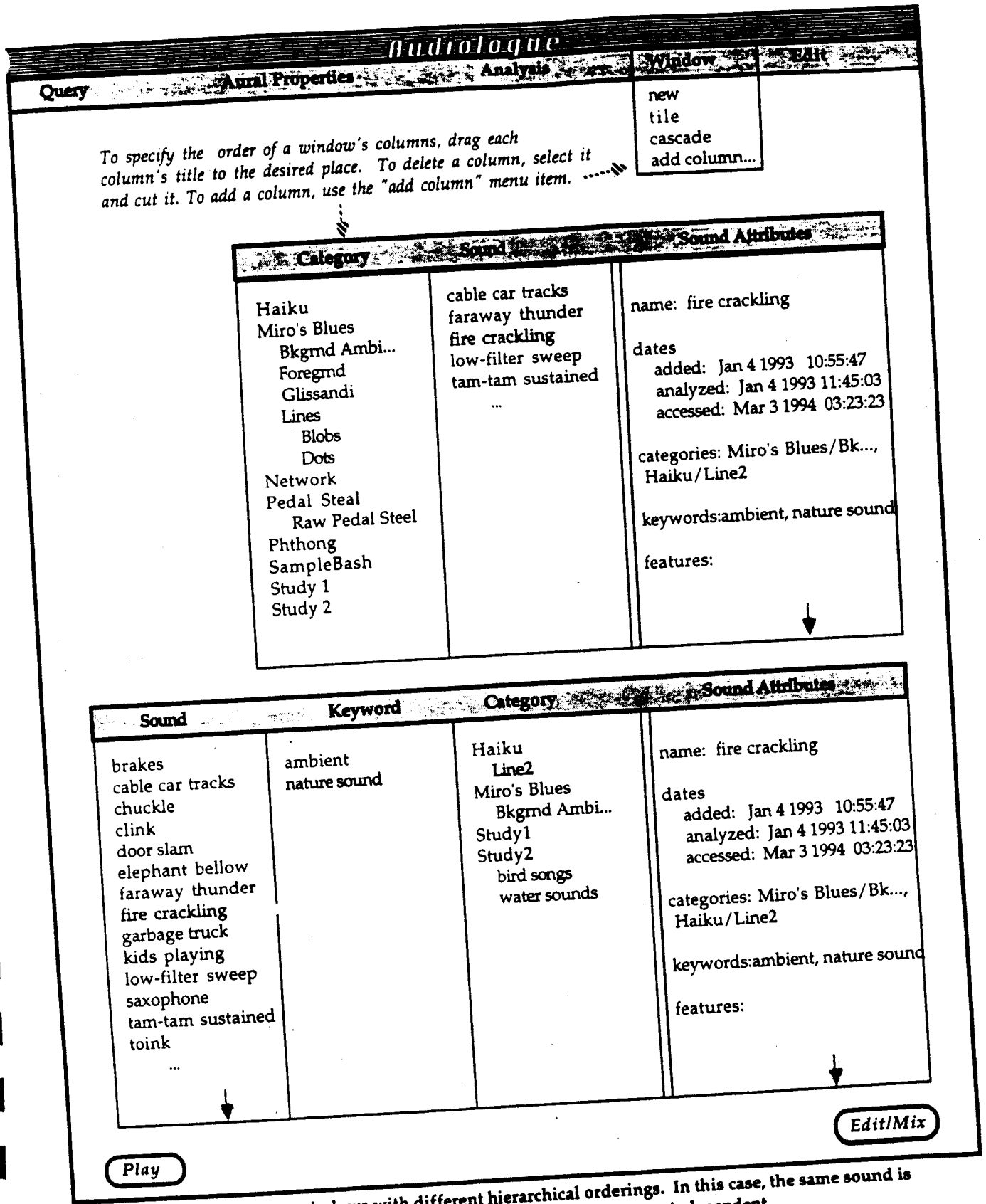


Fig. 3 - Example of two list windows with different hierarchical orderings. In this case, the same sound is shown in both windows, but two different sounds could be: the windows are independent.

5.4 Aural Properties Menu

Figure 4 illustrates the use of the *Aural Properties* menu. The user is able to define new aural properties by choosing *new*. When setting up a new aural property, the user is prompted to choose between a continuous and a discrete property, as shown. (See section 3.3, above, for the mathematical treatment of these two types of property.) In either case, the user enters the name of the new property in the dialog that pops up; in the discrete case the user must also specify how many discrete values the property has. Then the user drags "training" sounds into the lists labeled "Example sounds." These sounds are typically dragged from a list window of the type shown in previous figures.

In the discrete case, the user specifies the value for the sound by dragging it to the correct list.

In the continuous case, the single list is ordered by value. An initial value is automatically assigned to a new sound by dividing the existing range into equal increments and giving the new sound a value corresponding to the position where it was dropped. The user can audition sounds by double-clicking them, and can drag sounds to a different position to reorder the list, automatically creating new values. For greater control, the user can explicitly type in values (or use the value slider to set the selected value). A value that has been explicitly set will not be automatically changed when the list is reordered or extended.

At any point during the assembly of the example sound list, the user can request the system to analyze the aural property, by clicking the *See Analysis* button. This brings up another window (not shown) in which the user can inspect the existing analysis, if any has been done, or initiate the analysis. When the user saves the property (using the *save* menu command), the system schedules the analysis of all sounds in the database, which are assigned values of the property that depend on their measured attributes.

An aural property that has already been defined can be inspected and modified by choosing *open*, which brings up the same sort of panel that was used to initially define the property.

The job of defining new aural properties should not be taken lightly in a multi-user environment, since it modifies every record in the database. It would probably make sense to restrict access to this menu or to issue warnings when it is used.

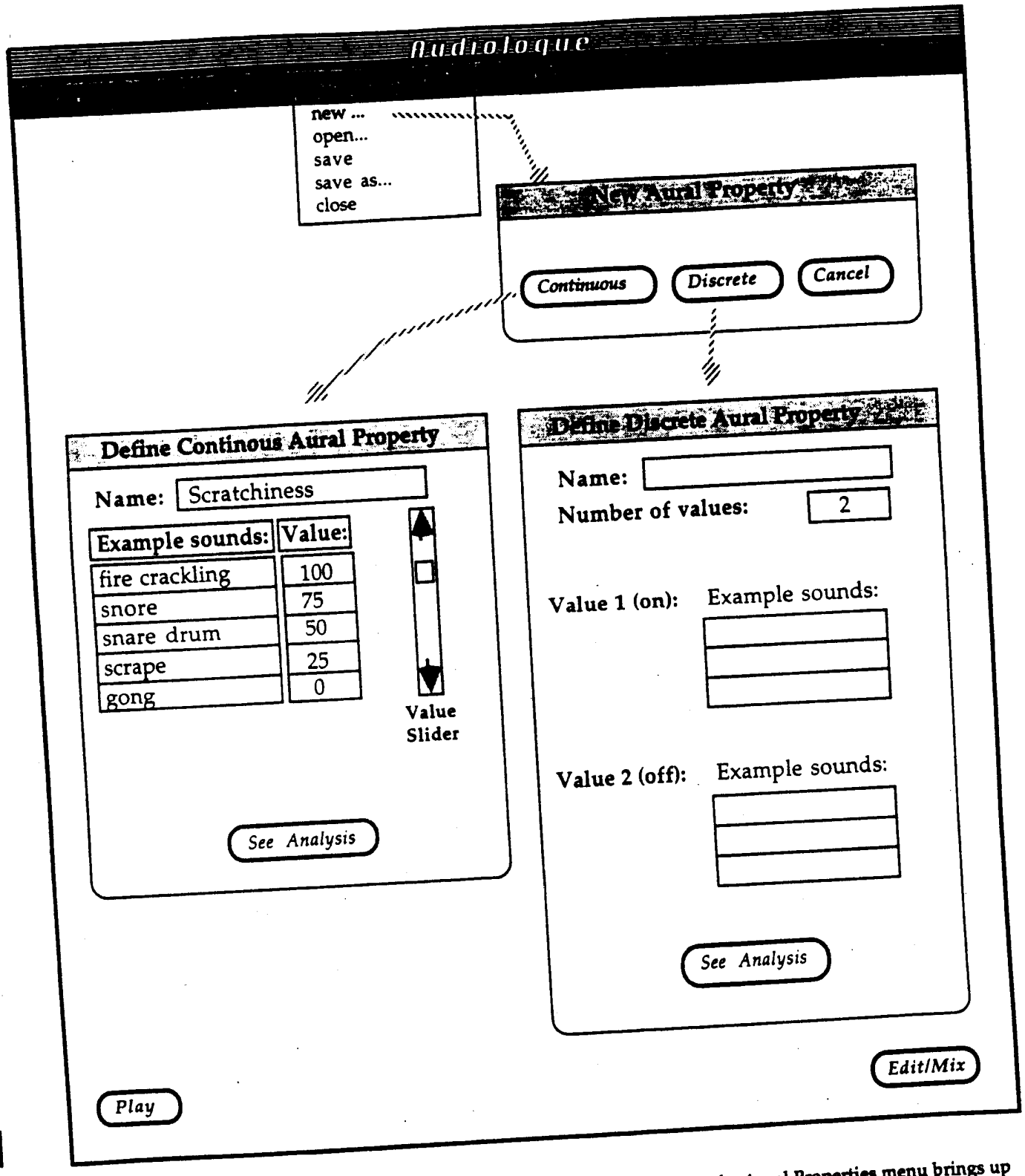


Fig. 4 - Defining a new aural property (subjective feature). Choosing "new..." in the Aural Properties menu brings up the New Aural Property window, in which the user can select either a continuous or a discrete property. Each choice brings up a different dialog, as shown. The user can drag sounds from a main multi-list window into any of the "Example sounds" lists.

5.5 Analysis Menu

The Analysis menu, pictured in Figure 2, contains a command to view the analysis results and another command to schedule when analyses should take place. The former command brings up a panel (not illustrated) from which the user selects which parameters to view. The latter command is described below.

The sound analysis procedures are time-consuming. Therefore, various schemes exist for telling the system when it should take the time to analyze sounds that have been added to the database. Analysis-scheduling options include:

• *Automatic* . The application triggers the analysis of a new sound immediately upon its entry into the database.

• *On Reference*. The application triggers the analysis of a sound the first time it is explicitly referenced—for example, when a user selects a specific sound and asks the system to find sounds that are similar to it. In queries, only sounds which have been analyzed by the system are candidates in the search (if that search involves feature-list comparisons).

• *When Idle*. The application uses its own idle-time to trigger analyses of sounds in the database. The schema contains flags which are used to determine which sounds have not yet been analyzed.

• *At Time* . The application enables the user to schedule dates and times for the database analysis procedures. The user can specify that these time-consuming procedures run during convenient, low-load hours.

5.6 Query Menu

Figs. 5 and 6 demonstrate the query operations. The user constructs queries by specifying either search values or sounds. Figure 5 shows the *Query->query by value* technique. Here the user has specified a variety of search criteria including keywords, high-level perceptual features, and low-level acoustic attributes. (Almost any data field can be used as a criterion.) The criteria within and across the levels of description can be constrained using boolean operators such as *and* (&&), *or* (||), and *exclusive-or* (^). Arithmetic operators (less-than, greater-than, etc.) can be used where appropriate. The query by value method looks for static (as opposed to relative) membership—that is, it looks for sounds whose attribute values match or fall within the ranges of values specified in the search criteria. The search is confined by the current selection (if any) in the list window. For example, if one or more categories are currently selected in the Category column, only sounds in those categories are searched. If nothing has been selected in advance of the query operation, the system will test all sounds in the database. This feature can save time when constructing queries.

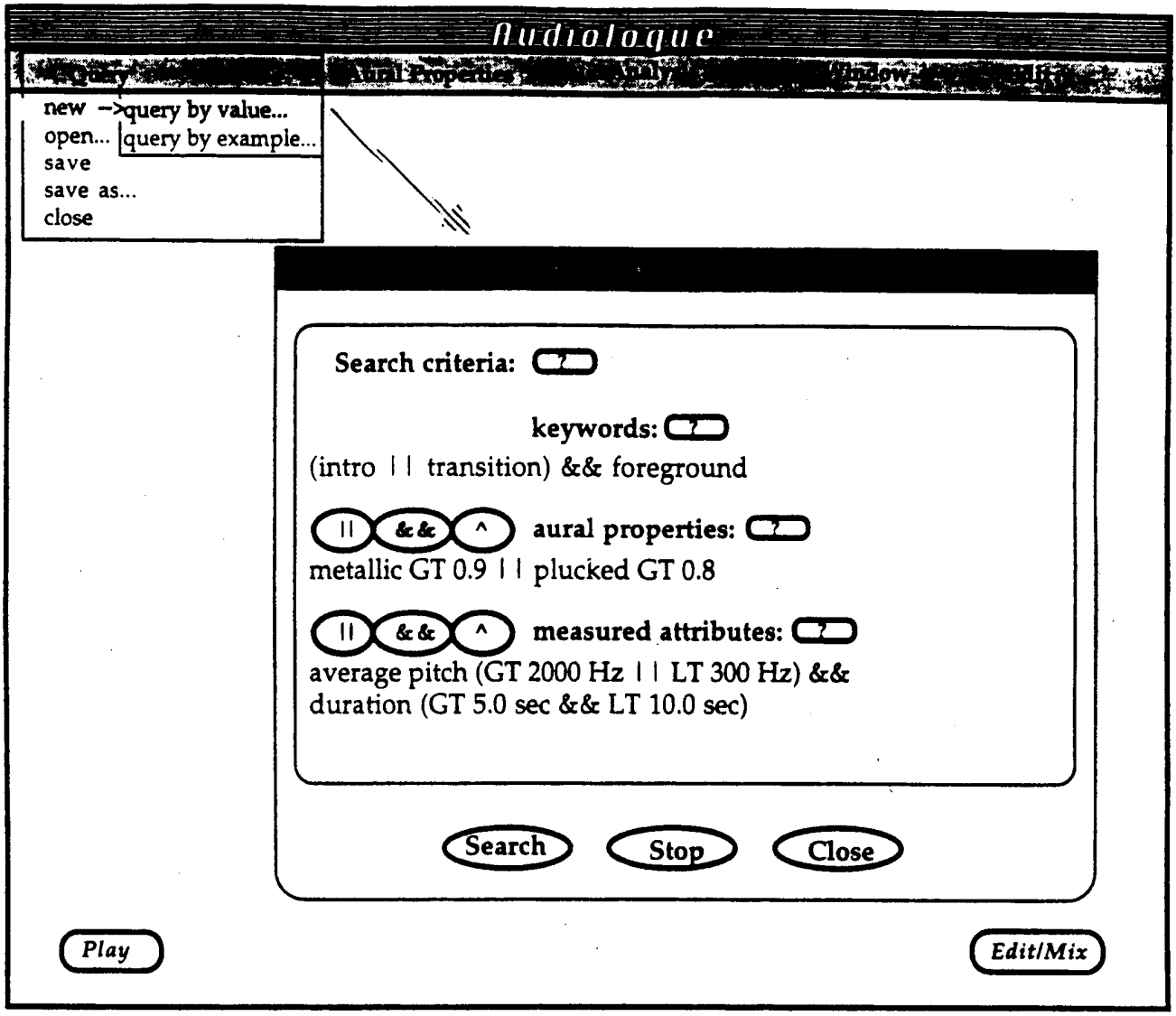


Fig. 5 - Query by value. The user clicked the "?" button to the right of the label "Search criteria:" and selected "keywords," "aural properties," and "measured attributes" from an ensuing "stay-put" list (not shown) of all known attributes. Each selected attribute is displayed below with its own "?" button for choosing or clearing values of the attribute and for choosing Boolean operations on the values. The chosen criteria (keywords, aural properties, and measured attributes in this example) can be themselves be connected with Boolean operations, using the buttons to the left of the attribute name. The Stop button halts an in-progress search.

Figure 6 shows the *Query-> query by example* technique. Here the system will test for proximity to a selected source sound: it will search for sounds whose features (measured attributes and aural properties) have values similar to those of the user-selected (source) sound. (This proximity rating is based on a metric, not described here, that scales the values for the various parameters depending on their psychoacoustic characteristics.) One can think of this query as a relative (as opposed to a static) search. The *Query by Example* dialog contains settings for "very similar," "somewhat similar," and "very dissimilar." (It might be desirable to let the user specify arbitrary amounts of similarity.) The source sound for the search should be selected by the user prior to invoking the *Query->query by example* operation. If the user omits this step, the *Query by Example* dialog will issue a prompt. The source sound may be any sound in the database, an accessible sound file which is outside the database, or a sound read in through the machine's sound port. Only sounds that have been analyzed are tested (compared) during any search operation. If the user has configured the system for automatic analysis (see Figure 2), any unanalyzed sounds within the search path will be processed. The query can be constrained to compare particular parameters: pitch, loudness, harmonicity, brightness, formants, and/or duration. Combinations of these constraints are possible. To "zoom" into a parameter for finer control, the user can double-click on the parameter's box to bring up the components of that parameter's trajectory (average, maximum, minimum, variance, autocorrelation, and shape parameters.)

Results for query by value and query by example are displayed identically. The results are displayed in a window that is almost the same as a regular multi-list window. The columns can be re-organized as usual, and items within a column can be sorted as usual. One can select multiple entries (sounds) within the results window and then invoke the *Query->query by example* operation again. This can be a useful means of refining a query. The *Play* and *Edit/Mix* buttons function on the currently selected sound(s), just as they would with a regular multi-list window. One can also select sounds in the results window and drag them to another window, such as the Aural Properties window or the Category column of a multi-list window.

The only difference between a Query Results window and a normal multi-list window is that the Query Results window includes a button called *see query*. This pops up the query window that generated the results window (in case the query window was obscured or closed), so that the user can modify the query.

It is easy to imagine browsers for other media (video, stills, animation) communicating search criteria to the audio browser. One can extend this concept, slightly, to envision a video clip browser that can automatically translate video search criteria into relevant search criteria for the sound browser. Effectively, through inter-application communication, the video browser would graze the sound-effects database and construct a mix-list of sounds to accompany a mix-list of video clips. This could be very useful for multimedia document authors, especially during stages of prototyping. The audio browser could similarly communicate with other application programs geared toward other usages (such as medical analyses).

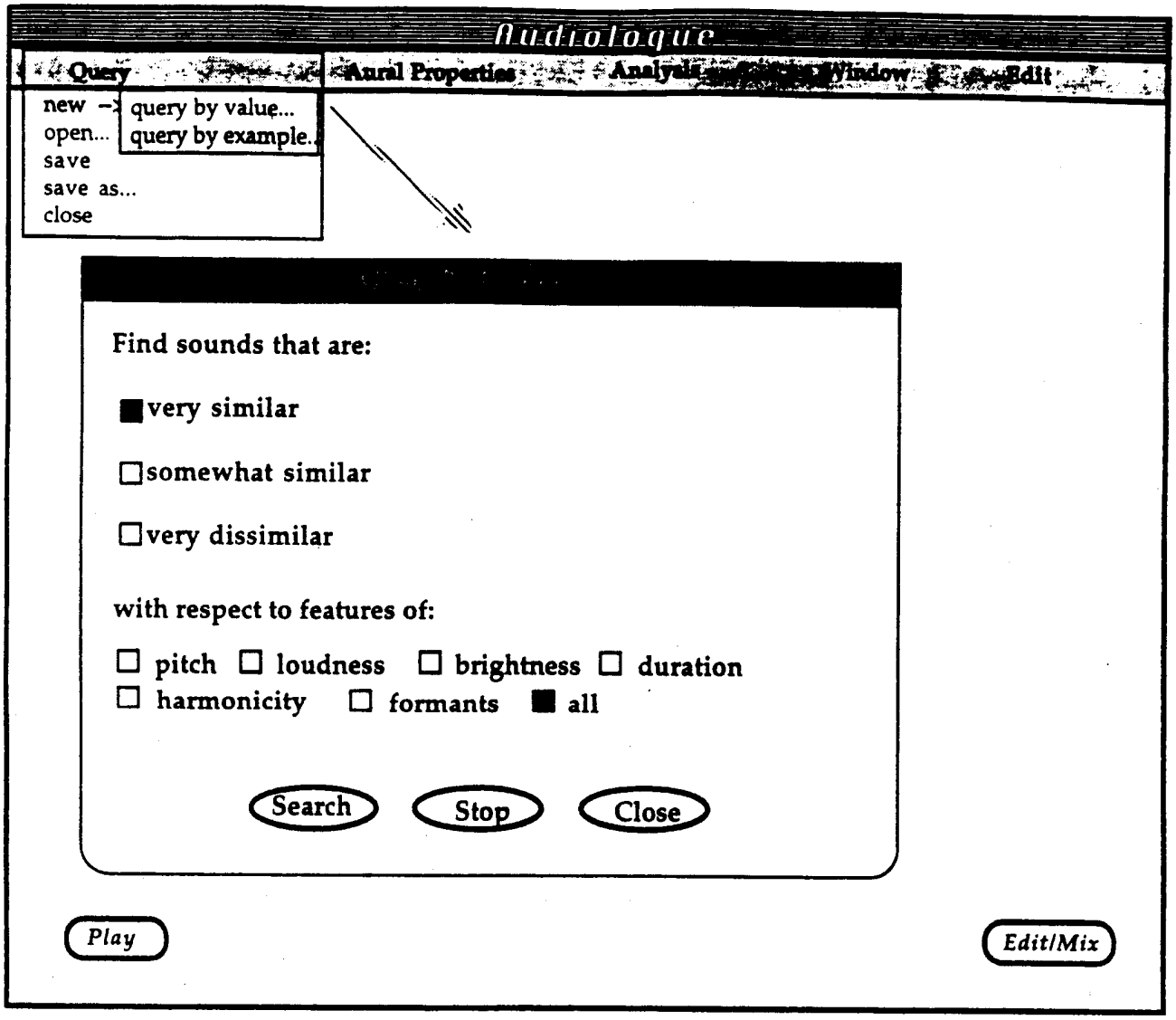


Fig. 6 - Query by example. User asks the system to produce a list of sounds that are very similar, somewhat similar, or very dissimilar to the source sound. The source sound is the single sound currently selected in the list window (not shown). The proximity (comparisons) can be made with respect to all the sound's measured attributes (the default), or with respect to duration or to a certain group of attributes (those related to pitch, loudness, or spectrum).

5. Conclusion

This paper has outlined some of the main features of a proposed analysis engine and front-end application for content-based retrieval of audio.

There are many other possible features and approaches for such a system. For example, it might be desirable to allow the analysis engine to operate on sounds not stored in the database. In this way, a user could acoustically analyze a sound file prior to entering it in the database, in order to determine its suitability.

As another example, a supplementary sound synthesis feature could assist the user in making queries. When the user was unsure what values to use, the synthesis feature could create sound examples using different user-specified values. The user could approximate the desired sound by testing different values for each of the measured attributes, refining the synthesized example until it bore enough similarity to the desired sort of sound. (The synthesis might also include signal processing of stored sounds, so that the user could select an existing sound as a starting-point for refinement.)

As mentioned earlier, this is work in progress. Further implementation and testing of our system will reveal whether the chosen acoustical attributes are sufficient (or excessive) for usefully analyzing and classifying most "atomic" sounds (i.e., brief sound effects). Practice with the system will also undoubtedly suggest various modifications and refinements to the prototype user interface. We believe, however, that the basic approach presented here is feasible for a wide variety of audio database applications.

References

Ackerman, D. 1991. *A Natural History of the Senses.* New York: Vintage Books. 173-225.

Blum, T. 1982. "Phthong -- An Interactive System for Music Composition." In *Proceedings of the 1982 International Computer Music Conference.* San Francisco: International Computer Music Association.

Bolinger, D. 1975. *Aspects of Language.* 2nd ed. New York: Harcourt Brace Jovanovich, Inc.

Bowen, T. F., G. Gopal, G. Herman, T. Hickey, K. C. Lee, W. H. Mansfield, J. Raitz, A. Weinrib. 1992. "The Datacycle Architecture." In *Communications of the ACM* 35(12): 71-81.

Bregman, A. 1993. "Auditory Scene Analysis: Hearing in Complex Environments." In *Thinking in Sound: The Cognitive Psychology of Human Audition,* S. McAdams and E. Bigand. Oxford: Clarendon Press.

Buxton, W., S. Patel, W. Reeves, and R. Baecker. 1981. "Scope in Interactive Score Editors." In *Computer Music Journal* 5(3): 50-56.

Buxton, W., S. Patel, W. Reeves, and R. Baecker. 1982. "*Objed* and the Design of Timbral Resources." In *Computer Music Journal* 6(2): 32-44.

Chafe, C., D. Jaffe, K. Kashima, B. Mont-Reynaud, and J. Smith. 1985. "Techniques for Note Identification in Polyphonic Music." In *Proceedings of the 1985 International Computer Music Conference.* San Francisco: International Computer Music Association.

Chafe, C., B. Mont-Reynaud, L. Rush. 1982. "Toward an Intelligent Editor of Digital Audio: Recognition of Musical Constructs." In *Computer Music Journal* 6(1): 30-41.

Charbonneau, G. R. 1981. "Timbre and the Perceptual Effects of Three Types of Data Reduction." In *Computer Music Journal* 5(2): 10-19.

Depalle, Ph., G. Garcia, X. Rodet. 1993. "Analysis of Sound for Additive Synthesis: Tracking of Partials Using Hidden Markov Models." In *Proceedings of the 1993 International Computer Music Conference.* San Francisco: International Computer Music Association.

Desain, P., and S. de Vos. 1992. "Autocorrelation and the Study of Musical Expression." In *Music, mind and machine: studies in computer music, music and artificial intelligence,* P. Desain and H. Honing. Amsterdam, The Netherlands: Thesis Publishers. - Ill.

Eaglestone, B. 1988. "A Database Environment for Musician-Machine Interaction Experimentation. In *Proceedings of the 1988 International Computer Music Conference.* San Francisco: International Computer Music Association.

Foster, S., W. Schloss, and A. J. Rockmore. 1982. "Towards an Intelligent Editor of Digital Audio: Signal Processing Methods." In *Computer Music Journal* 6(1): 42-51.

Freed, A., X. Rodet, and Ph. Depalle. 1993. "Synthesis and Control of Hundreds of Sinusoidal Partials on a Desktop Computer without Custom Hardware." In *Proceedings of the 1993 International Computer Music Conference.* San Francisco: International Computer Music Association.

Grey, J. M. 1977. "Multidimensional Perceptual Scaling of Musical Timbres." In *Journal of the Acoustical Society of America* 61:1270-1277.

Gordon., J.W., and J. M. Grey. 1978. "Perception of Spectral Modifications on Orchestral Instrument Tones." In *Computer Music Journal* 2(1): 24-31.

Handel, S. 1989. *Listening: an introduction to the perception of auditory events.* Cambridge, Massachusetts. M.I.T. Press.

Helmholtz, H. 1885. *On the Sensations of Tone as a Physiological Basis for the Theory of Music.* Trans. Alexander Ellis. Reprint New York: Dover, 1954.

McAdams, S. 1984. *Spectral Fusion, Spectral Parsing, and the Formation of Auditory Images.* Diss., Stanford University.

McAdams, S. 1993. "Recognition of Sound Sources and Events." In *Thinking in Sound: The Cognitive Psychology of Human Audition,* S. McAdams and E. Bigand. Oxford: Clarendon Press.

Minsky, M. 1981. "Music, Mind, and Meaning." In *Computer Music Journal* 5(3): 28-44.

Moorer, J. A. 1975. "On the Segmentation and Analysis of Continuous Musical Sound." Report STAN-M-3. Stanford: Stanford University Department of Music.

Moorer, J. A. 1977. "On the Transcription of Musical Sound by Computer." In *Computer Music Journal* 1(4): 32-38.

Moorer, J. A., J. Grey, and J. Strawn. 1977. "Lexicon of Analyzed Tones -- Part 2." In *Computer Music Journal* 1(3): 12-29.

Moorer, J. A., J. Grey, and J. Strawn. 1978. "Lexicon of Analyzed Tones -- Part 3." In *Computer Music Journal* 2(2): 23-31.

Plomp , R. 1976. *Aspects of Tone Sensation: A Psychophysical Study.* London: Academic Press.

Risset, J.-C. and M. Mathews. 1969. "Analysis of Musical Instrument Tones." *Physics Today* 22(2):23-30.

Roads, C. 1982. "A Conversation with James A. Moorer." In *Computer Music Journal* 6(4): 10-21.

Roads, C. 1983. "A Report on SPIRE: An Interactive Audio Processing Environment." In *Computer Music Journal* 7(2): 70-74.

Schaeffer, P. 1977. *Traité des objets musicaux.* 2nd ed. Paris: Seuil.

Schafer, R. M. 1980. *The Tuning of the World: Toward a Theory of Soundscape Design.* Philadelphia, Pennsylvania: University of Pennsylvania Press.

Strong, W. J. and G. R. Plitnik. 1992. *Music, Speech, Audio.* Provo, Utah. Soundprint Publishing.

Tenney, J. 1961. "Meta-Hodos: a phenomenology of 20th-Century musical materials and an approach to the study of form." Original Master's thesis published by the Inter-American Institute for Musical Research, Tulane University, New Orleans (1964). 2nd ed. 1988. Oakland, California: Frog Peak Music.

Vertegaal, R. and E. Bonis. 1994. "ISEE: An Intuitive Sound Editing Environment." In *Computer Music Journal* 18(2): 21-29.

Wang, A. 1994. *Instantaneous and Frequency-Warped Signal Processing Techniques for Audio Source Separation.* Diss., Stanford University.

Wessel, D. L. 1979. "Timbre Space as a Musical Control Structure." In *Computer Music Journal* 3(2): 45-52.

Item -9-

The Networked Video Jukebox

Laurence Crutcher, *Member, IEEE*, and John Grinham

Abstract—**The availability of low-cost compression/decompression (codec) hardware for video and audio has given us the opportunity to provide multi-media applications over data networks. This paper describes a particular system that we have built to demonstrate this capability. A video jukebox has been built using off-the-shelf computers connected over Ethernet. For wide area connectivity, SMDS attachment has been used. Combined video and audio clips can be selectively played on a PC equipped with low cost video and audio codec cards, using JPEG and ADPCM compression hardware. The data for these clips is retrieved in real time from a Unix workstation via the network. Following a description of the system design, the performance obtainable from a number of network configurations is shown from the results of the experiments we have conducted. These results show that the addition of real-time transport software on top of the standard Internet protocols allows us to provide video and audio clips with a system comprising standard off-the-shelf equipment. The quality of these clips is considered acceptable for many applications. We have quantified the capacity of the system in terms of the data rates and the number of users, and have identified which parts of the total system become bottlenecks as each of these parameters is increased. Solutions to overcoming these bottlenecks are described.**

I. Introduction

THE availability of low-cost compression/decompression (codec) hardware for video and audio has given us the capability to provide multi-media applications over data networks. Data networks that can support the reduced bandwidth of compressed video and audio are in place and readily available. Powerful desk-top computers that can process this data are rapidly advancing to the point where they are commodity items. Together, these technologies present an exciting opportunity to provide applications that utilize video and audio to users now.

The acceptance of new applications employing multi-media is dependent on a balance between price and performance. It is therefore appropriate to ask just what is achievable with the technology that is available, and to attempt to identify limitations in systems built with this technology. Further, we should quantify how the capabilities of a system increase as we make advances in each of the limiting areas.

This paper adopts this approach, using the design and performance of a networked video jukebox that we have built as the basis of our study. The video jukebox allows a user to view video clips, with stereo audio, on a PC equipped with low cost video and audio codec cards, using JPEG and ADPCM compression hardware. The jukebox is operated from a Windows application that presents the user with a control panel of the type used for remote control of a domestic video recorder. The user can view a list of the clips available and control the display of a clip through the selection of the appropriate buttons on this panel. All of the data for the clips is stored on the hard disk of a remote server with which the client communicates over an intervening network. The server is a Unix workstation. Applications for this specific type of system include training, education and video mail.

The prototype system uses Ethernet to connect the client and server over the local area. Experiments have also been conducted with the client and server on separate LANs interconnected via a T1 SMDS link [1]. The operation of the video jukebox over networks with modest bandwidth is enabled by the use of compressed video and audio. The PC contains two cards that provide compression and decompression of video and audio in real time. Thus the PC is used to record clips, which are then transferred to the server for storage. As the jukebox client, the same PC decompresses the clip during play back.

The motivation for the work reported here is to examine the requirements and possible solutions for networked multi-media applications from a systems perspective. The video jukebox is an example of a specific class of application that focuses this study. Previous articles in the literature have looked at similar systems from either a qualitative [2] or theoretical [3] point of view. This paper complements that work with a study based on measurements from a working system.

The components of the system can be categorized into the following areas.

- Computers for the client and server.
- Video and audio subsystems.
- Network connecting client and server.
- Network services.
- Application software.

In building and testing the video jukebox, our aim is to isolate the particular problems presented by each of these components, and to identify possible solutions. The main objective is to use off-the-shelf equipment wherever possible, and to demonstrate where the limitations are with this approach. We have quantified the capacity of the system in terms of the data rates and the number of users, and have identified which parts of the total system become bottlenecks as each of these parameters is increased. Solutions to overcoming these bottlenecks are described.

We have shown that a working system can be built using off-the-shelf parts, together with some software for end-to-end protocols. This system provides video and audio with a quality

Manuscript received July 1, 1992; revised August 23, 1993.

L. Crutcher is with the Market Vision Corp., 40 Rector Street, New York, NY 10006. This work was carried out as an employee of Hewlett Packard.

J. Grinham is with Hewlett Packard Laboratories, Bristol BS12 6QZ, England.

This manuscript was recommended by Ali Tabatabai.

IEEE Log Number 9215864.

considered acceptable for many applications. Specifically, we have made the following measurements.

- A standard 386 PC client, equipped with video and audio hardware, connected to a standard HP series 400 Unix workstation configured as a video and audio file server, is sufficient to support a compressed video stream of 256 × 160 pixels at 20 frames/sec in full color alongside a stereo audio channel of 4 bit ADPCM. The bit rate of such a stream is ~ 0.8 MBit/s.
- Up to 8 such streams could be supported on a single 802.3 network dedicated to this application, without noticeable degradation of the image or audio quality, and in addition a single stream can be supported over an SMDS WAN using the current prototype HP 802.3-SMDS router.
- The number of clients that can be served simultaneously by a server is dependent on disk performance and the distribution of data blocks over the disk. In the worst case, without imposing any constraints on this distribution or on the scheduling policy in the file system, the HP 400 workstation used in our experiment could support about 8 clients, each reading a separate video and audio clip. At the time of writing, the latest disk technology could be expected to double this number.

Our general observations on how to extend networked multimedia systems to handle higher data rates, more clients, and other classes of applications, are described in the concluding section of this paper.

In the next section of this paper, an overview of the system architecture is given which covers the configuration of the components and the video and audio subsystem. The following section then gives an analysis of the performance of the system components and defines the limits within which the system will operate. The network services and software design are described in Section IV. Section V shows the results of the measurements that have been made on the system in different configurations and Section VI discusses the implications of these results. The paper is concluded in the last section.

II. System Architecture

1. Hardware Configuration

The hardware consists of a client HP Vectra RS25/C PC equipped with prototype video and audio decompression hardware[1], connected via standard network interfaces to an HP 9000/400 series workstation server (Fig. 1).

The most simple network configuration for connecting the client and the server used a private Ethernet LAN. The only other station on this LAN was a network monitoring tool which allowed detailed network traffic measurements to be made. Further experiments were conducted with the system in the presence of other LAN traffic and also with a connection over an SMDS wide area network. This last configuration is shown in Fig. 2.

1.A. *Video Subsystem*

The Videologic prototype video codec was built around a C[3] Microsystems JPEG compression/decompression chip

[1] Manufactured by Videologic Limited.

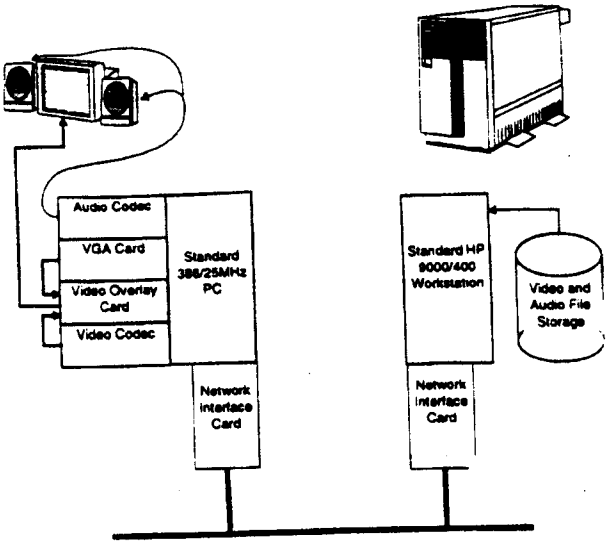


Fig. 1. Video Jukebox Hardware Configuration.

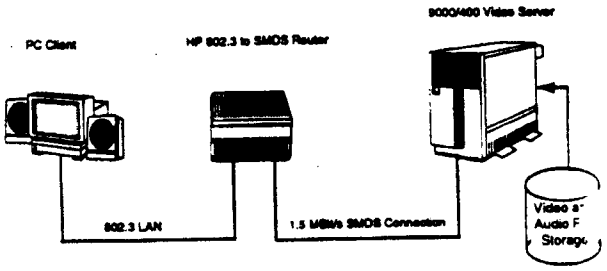
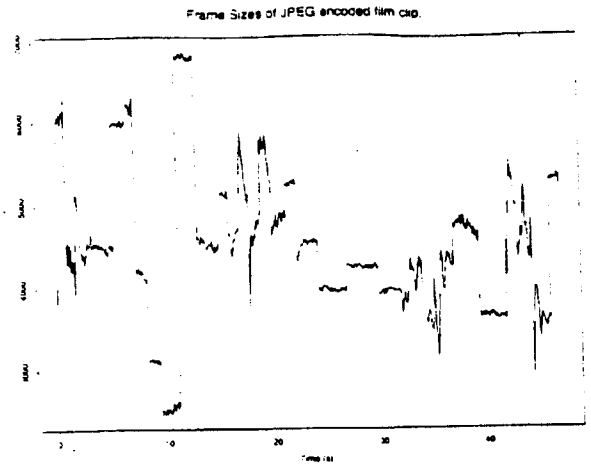


Fig. 2. Wide Area Network Configuration.

controlled by an on card transputer together with a 256 KByte FIFO for incoming and outgoing data storage. The transputer controls the interface to the host PC, the setting up of the compression chip, and the management of the FIFO. The prototype transputer code allows the user to select the image size to be coded together with the frame rate and quantization levels.

The JPEG standard is aimed at continuous tone still image compression so there is no inter-frame compression. The addition of inter-frame compression such as in the MPEG standard would increase the compression by a factor of about 3. While MPEG is targeted at 1.5 Mb/s, there is no inherent reason that the standard could not be used at the lower bit rates used in these experiments to obtain higher quality images and audio. (For a detailed description of JPEG and MPEG, see the articles in [4].)

Fig. 3 shows the JPEG-coded frame sizes against time for the trailer sequence of the film 'Buster' which was used in the experimental sections of this work. The scene changes in this sequence can be clearly seen. Typical variations with JPEG on this type of material are of the order 3:1, considerably smaller than the 10:1 variations which could be expected using JPEG. It is important to note that different image sequences would



F 3. JPEG Compressed Frame Sizes.

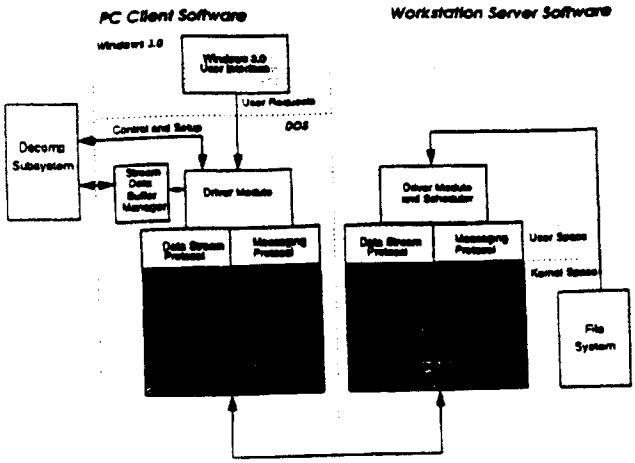


Fig. 4. Video Jukebox Software Architecture.

have quite different characteristics. A sequence that consisted of a long 'talking-head' shot followed by a steady text frame would effectively be at two distinct data rates.

1.B. *Audio Subsystem*

The audio codec is a general purpose DSP chip which is programmed to implement two channels of 4 bit ADPCM compression on 16 bit input samples at sampling rates of either 9.45 kHz, 18.9 kHz or 37.8 kHz. This card also has a controlling transputer and a 64 Kbyte FIFO which corresponds to around 6 seconds of audio at a sample rate of 9.45 kHz and 8 bits per stereo sample.

In our experiments we used stereo 4 bit ADPCM at the lowest sample rate and played the audio back through small single speakers. While this is a highly subjective result, we judged the quality to be similar to high quality AM radio, and quite effective in stereo for replaying a film soundtrack.

C. *Video and Audio Synchronization*

VideoLogic's prototype video and audio system was originally designed to store and replay video and audio from a local disk. The modifications necessary to run the system using a remote filestore were carried out at Hewlett-Packard Labs by the authors. In Videologics original prototype stand-alone system there is no explicit synchronization recovery mechanism. The compressed audio and video data are stored in separate files. A third file is created during recording and used for two functions. The first is to index on a frame by frame basis into the video data file. This is required because of the variable compressed frame length. The second function is to retain a list of frame display times associated with each frame. This is required if, due to host system overheads, it is not possible to continuously transfer data to the local disc at the required rate so that recorded frames are no longer compressed at the nominal rate.

To play back a clip in the stand-alone system, the decompression subsystem first opens the locally stored files, gets the audio and video parameters, and then displays the first video frame in the clip. It then requests a block of audio data samples, queues them in its local FIFO and starts playing them back. Requests are then issued for video frames from the local file system. These are obtained together with the display time from the index file. The frame and its display time are transferred, queued in the local FIFO, and then displayed for the required amount of time.

This scheme inherently makes the assumption that no frames are lost between recording and playback. If they are then synchronization between the audio and video channel is lost. The implications of this in a networked environment are discussed in Section VI.

1.D. *Video Overlay and Capture*

The video overlay card is connected to the compression card via a digital bus that is closely related to the CCIR 601 digital video standard. Using this bus decompressed video from the compression card can be chromakeyed into a display window or alternatively analog video can be frame grabbed, digitized, and fed to the compression card.

When in capture mode the image size can be scaled before grabbing to a variety of sizes from 192 × 128 pixels to 352 × 256. We mostly used 256 × 160 which occupies roughly a quarter of a VGA screen. During playback it is possible using this card to expand the picture so that the original 256 × 160 image can be blown up to 484 × 320 pixels, which is roughly 50% of the graphics area. With the very small viewing distances that are used with computer displays this picture zooming is not particularly useful—the image defects are highlighted and are very noticeable. However, at viewing distances more closely associated with television the picture looks similar in quality to VHS video.

2. *Software Configuration*

The software architecture is illustrated in Fig. 4. The PC software runs under Windows 3.0a running in enhanced mode. The top level user interface program is a windows program that resembles a video recorder remote control. Menus allow dialog boxes to be pulled up which in turn allow a video clip to be selected and played. Control messages from the user interface are passed to the main system driver module which runs as a DOS virtual machine.

This software architecture takes advantage of the ability of Windows 3.0a to multitask between the Windows programs and any number of DOS virtual machines that are running. The driver module, which contains all the protocol processing and is also responsible for the data transfer of the video and audio streams from the network to the decompression hardware, can thus be allocated a large number of time-slices compared to the windows program. The effect of other system activity such as movement of windows or the mouse can therefore be minimized. This is discussed in more detail in Section III.

The driver module takes messages from the user interface program and calls the top-level protocol functions as described in Section III. Under normal circumstances the control protocol will complete correctly and the driver module will be able to read audio and video data from the top level of the data stream protocol. The control and data protocols are built directly on top of HP's DOS Arpa sockets product. This provides Berkeley sockets network services on a PC.

Transfer of the video and audio data to the decompression subsystem from the network subsystem is done using a shared buffer area and a special token. The driver module requests a token from the decompression subsystem, requests buffer space from the buffer manager and then, if the decompressor has room for the data in its receive FIFO, it looks at the top of the data stream layer to see which type of data is waiting to be received. The data, which in the case of video must be a whole frame, is copied into the shared buffer and the token is handed back to the decompression subsystem with information on the type and amount of data in the shared buffer. This data is then asynchronously read, queued, decoded, and displayed by the decompression subsystem.

The server software operates in user space and, similarly to the PC, the underlying communications software is HP's Berkeley sockets implementation. The video and audio files are stored on the workstations internal disk as part of its normal file system. Storage on to a raw disk partition has not been used in this implementation.

III. Analysis

1. Overview

The basic premise of the networked video server is that the isochronous video and audio data can be read from a remote disk and transferred to the client over an existing computer data network at a controlled rate. The important factors in determining the perceived audio and video quality are the mean data rate and the data loss rate. Data loss in this context effectively includes data that arrives too late or out of order.

Data loss can be caused both by bit errors on the network and by overflowing or emptying of data buffers. Data loss on the audio channel results in very noticeable clicks and pops. Fortunately, even though the coding is essentially differential on a sample by sample basis, the data is packetised into self contained chunks which can be played without reference to previous data. Thus loss of a packet does not cause all the following data to be unplayable.

Data loss on the video channel is not so instantly noticeable. At 20–25 frames/second it is quite possible to lose a frame without the viewer noticing. However, since there is no explicit time synchronization between the two channels, data loss gradually degrades the lip synchronization. Lip synchronization effects become noticeable when the timing of the sound relative to the video exceeds approximately −40 ms to +20 ms [5], but its importance is highly dependent on the material being viewed and the susceptibility of the viewer.

The mean rate and delay variance at the receiver must therefore be controlled sufficiently accurately to ensure that th probability of the FIFOs on the compression cards overflowin or emptying is sufficiently low. On the system we used, if the audio FIFO empties then both the audio and video replay stops until more audio data arrives. If the FIFOs become full the compression subsystem no longer requests data from the host machine. Since there is no packet by packet feedback to the server, more data arrives on the network and has to be queued in the network subsystem. The queue lengths available in the network subsystem are quite small (around 10–20 KBytes) a so data is soon discarded resulting in a very high data los

The variance of the data arrival at the compression subsystem is set by a combination of five processes:

- Server data read function.
- Server data transfer function to its network subsystem.
- Network access delay.
- Client network read function.
- Client data transfer function to the decompression subsystem.

The significance of each of these processes is assesse the following sections.

2. Server Data Read Function

On the server the system clock can be used to control the read and data transfer processes to try and maintain an isochronous data transfer to the network subsystem. The processes were run in user space on a standard Unix workst so the accuracy with which this can be done depends o clock itself and on any other operating system activity.

The mechanism used to read the system clock on the server gave a clock resolution of 0.1 milliseconds. This effectively sets the lowest limit on the isochronicity of the server read and transfer process.

The operating system activities can be divided into two components. Those directly related to the user process, specifically data caching and read time on the file system, and entirely independent of the user process.

Data caching by the kernel causes data to be read from the file system in file system block size units and stored in memory. As a result file system reads at contiguous locations within a file will take a variable amount of time depending on whether all the data is in the cache already and how much extra data is being read. The caching algorithm employed on HP-UX 8.0 is quite simple: when the last fragment in a block is read as part of a sequential read, the next block of file is also read into the buffer cache. So the last read on a block incurs the overhead of reading the next complete block

The read time on the file system is dependent primarily on the disk performance. Disk seek times at around 16–17 ms plus 8 ms of rotational latency for 5.25" disks are available on Hewlett-Packard's current workstations and some improvements can be expected with smaller disks. The seek, settling, and rotational delays significantly reduce the disk performance if multiple streams are read randomly off the same disk. In the worst case if the block size is 8 Kbytes and successive blocks are distributed across the disk then the throughput of a 4 MByte/s 5.25" disk falls to ~ 300 KByte/s, which would only be enough to support a couple of 1 MBit/s isochronous streams. Increasing the block size to 32 KBytes increases the throughput to around 1 Mbyte/s, enough for 8 independent streams. A disk rated at 10 MByte/s with a total seek and rotational latency of around 12 ms, transferring random 32 KByte blocks would have a throughput of around 2 Mbyte/s, sufficient for 16 independent 1 Mbit/s streams. It should be noted that these figures are for worst case distributions of data blocks.

The application independent factors affecting the server data read process can be minimized by leaving only the basic system processes running and ensuring that there is sufficient memory available on the machine to avoid paging. As the experimental results will show, these processes are negligible when compared to the disk performance.

3. Server Data Transfer Function

The server data transfer function is affected by the same independent operating system factors as the read process. The server dependent activities are as follows:

- The data copy from the user buffer to a kernel buffer.
- The protocol processing to convert the user data into a packet suitable for transfer to the network card.
- The queueing time and copying time from the kernel buffer to the network card.

Variable delay can occur in the execution time of the send call if either an independent system process is scheduled or if the network card has insufficient buffers available to allow the transfer. The latter will only occur if previous packets are still queued in the card because of excessive network access delays or if the required peak data transfer rate cannot be supported by the card.

4. Network Access Delay

The local network available to us for this work was Ethernet (others are considered in Section IV). For experiments in the wide area we had access to a point-to-point 1.5 MBit/s SMDS link via a prototype gateway.

The video server could operate in one of three LAN environments. The first and simplest would be a private network with only one server and multiple clients receiving the same data. The second case would be with two or more servers on a private network supporting a number of clients. The final case would be a server or servers on a shared LAN co-existing with some unknown mixture of other applications.

With only a single transmitter on an Ethernet the network access delay is effectively zero. While this is not a particularly startling result it does serve to remind us that network protocols that allow multicast transmission, do not use acknowledgements, and do not congest the network with management activity, can enable a single server to broadcast video and audio streams to multiple clients with only the propagation delay to consider.

The behavior of the system with two servers on an Ethernet network depends to some extent on the design of the network interfaces. If the network interfaces assert an interrupt between the transmission of each packet then it is likely that the interrupt service time will be longer than the Ethernet interpacket time. Thus two servers would quickly synchronize: one defers to the other and then transmits without contention from the first. Thus the variability of the delay experienced by each packet would effectively be restricted to the maximal length packet transmit time of around 1.2 millisecond.

The situation becomes almost impossible to predict if the server(s) have to coexist on a shared LAN. Since Ethernet does not provide any MAC layer priorities or service guarantees it is not possible to separate out the isochronous data from the ordinary computer data. If any headway is to be made in characterizing the possible performance, some assumptions have to be made on all of the following:

- Distribution and number of machines on the network.
- Packet length distribution from each machine.
- Data load on each machine and the impact of higher level flow-control and buffering.

Some observations can be made. In many environments the long term average LAN utilization is of the order of just a few percent. Significant increases in the utilization occur during the course of the day where the minute by minute average might peak at 5–10%. In the very short term, i.e. on a second by second basis, very high peaks corresponding to file transfers may occur. In a laboratory environment our own measurements and those shown in [6] indicate that packet distributions are bimodal with the peaks towards the shortest and at the longest packet lengths.

For the video server the most important measure of the network is the variability of the packet delay. An extensive experimental study [7] conducted measurements on the standard deviation of the packet transmission delay for bimodal packet length distributions on a network comprising two sets of clustered hosts. This study indicates that for a balanced load the delay variation increases roughly linearly with the number of hosts going from 5 to 25. For a packet length distribution of 6 maximal length packets for every 2 minimal length and 20 hosts in 2 clusters on a 2000 foot Ethernet, the average transmission delay was 20 milliseconds with a standard deviation of 60 milliseconds. These delays were incurred with the ethernet utilization at 9.3 MBit/s. With just five hosts the average was around 5 milliseconds, with a standard deviation of 20 milliseconds at a similar average network utilization.

5. Client Network Read Process

The client network read process comprises the servicing of incoming packet interrupts from the network, the transfer of

the data to the host, and the protocol processing. The packet interrupt service time on the PC should be constant. The variability comes from the process of reading from the network subsystem (i.e. the top of the machine's protocol stack) and copying the data into application memory. The client machine is a PC running Windows 3.0a in enhanced mode. The network read process, and the data transfer process, are implemented in a DOS Virtual machine. In enhanced mode Windows time slices between the DOS virtual machines that are running and Windows itself. The length of the time-slice can be adjusted as can the effective number of time-slices that the virtual machine has. A balance needs to be struck between the throughput that the transfer process can achieve and the responsiveness of the top level windows application.

The variation in the delay incurred in the transfer process from the network card will be a function of the time-slice period. In our experiments we varied the time-slice between 10 and 60 milliseconds and changed the ratio of the number of time slices each process received from 1:1 to 5000:1.

6. Client Data Transfer Process

The client data transfer process takes data that has been read from the network subsystem and, in conjunction with the compression subsystem processor, transfers this data into the FIFOs on the compression cards. The basic mechanism is triggered by receiving data off the network. The data is copied, as described above, and the transfer process notifies the compression subsystem. If there is sufficient space on the card FIFOs a message is returned and transfer process copies the data in to a shared memory area on the PC. This data is then asynchronously collected and removed by the compression card processor.

The client process is part of the same virtual machine that is described above and is therefore time sliced. The collection of data by the compression card processor is interrupt driven and therefore should have relatively little impact on the variability of the data transfer delay.

7. Summary

From the above analysis only the following appear to be able to cause significant variation in the delay time of the data transferred from the server to the client:

- Server disk read times (5–25 ms variations).
- Network access delay on a very heavily loaded Ethernet (5–80 ms variations).
- Operating system time slicing on the PC (10–100 ms variations).

Thus the worst case variation in data arrival times that should be seen at the receive FIFOs on the decompression subsystem will be around 20–200 ms, depending on precisely how the system is set up and in what network environment it is used.

Since the distribution of actual delay times is unknown it is not possible to accurately calculate a given FIFO size for a given probability of the FIFO emptying or overfilling. However these figures imply that we should be looking to queue between 50 and 500 milliseconds of data depending on the environment in which we are operating. At a nominal data rate of 1 MBit/s this corresponds to roughly 6–60 Kbytes of data.

With the video jukebox application, a start up delay before playing a clip of 0.5–1 seconds is not objectionable. This should allow us to pre-fill the receive FIFO's sufficiently to smooth out the received data delay variations. In the experiments reported here the delay was set at 500 ms.

IV. Protocols

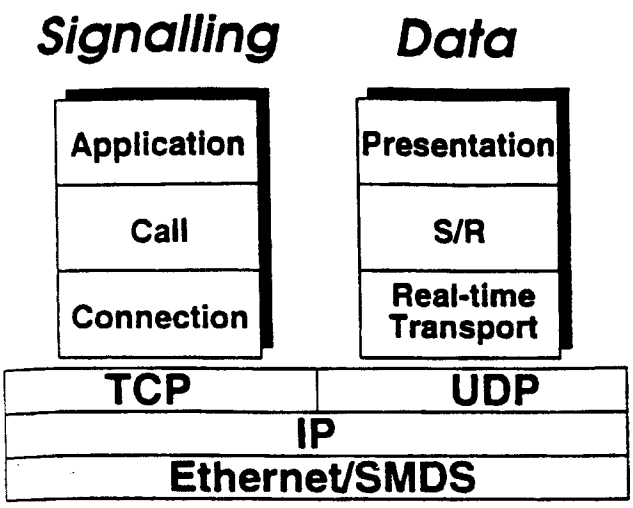
There are two distinct sets of requirements for protocols for multimedia applications. There is a need for signalling, both between hosts and within the network, and also data transport. End-to-end signalling allows hosts to exchange information for the control of the different components of an application, while host-network and intra-network signalling is required to establish and maintain the connections for information exchange. The requirements for data transport are the same as those for many other applications, such as file transfer or electronic mail, but there is an additional requirement to exchange data on a time scale that is determined by the application rather than by the capabilities of the underlying system.

In line with the philosophy of building a system based on off-the-shelf components, we have attempted to use existing protocols, as provided, where possible. However, there is currently no standard for signalling for multimedia systems, and no off-the-shelf implementation of a protocol for real-time data transport. In both cases, these are active areas of research and development. Signalling for multimedia services is addressed in [8] for example, while problems and solutions for connection management are described in [9]. For a discussion of the different approaches to real-time data transport, see [10]. For the system described in this paper, we have designed and implemented our own solutions for signalling and data transport. In each case, the protocols are implemented as software processes that run on top of standard Internet protocols, thereby maintaining a standard interface to the underlying network. The remainder of this section describes the design and implementation of these protocols.

1. Architecture

The protocol architecture comprises two stacks, the data stack and the signalling stack, as illustrated in Fig. 5. The data stack is responsible for sending and receiving data in real-time. The signalling stack is responsible for application level services and connection management.

For real-time data transport, a sending host transmits data at a rate that is specified by the application. This transport layer interfaces to the file system for presentation of data input. To optimally match the data rate and the size of the data units at the presentation layer to those of the underlying network, the segmentation layer may repackage the data where appropriate. A receiver is required to receive the data and to reassemble the data to reconstitute the original data structure. Here, the transport interfaces directly with the video and audio subsystem.



5. Protocol Architecture.

The signalling stack comprises the services for user interaction between the client and server, and those for call and connection management. The application services provide the remote interaction between the commands available to a user of the system and the corresponding operations on the server. Simple examples of these services are *play* and *stop*. The call management interfaces between the total network requirements of the application, and the underlying connection layer. The connection layer is responsible for the control of individual connections between the hosts.

The services provided in each stack use the standard IP protocol, together with either TCP or UDP, for the underlying network and transport services. The signalling stack requires reliable end-to-end communication and therefore uses TCP connections for the transfer of control information. For real-time data transfer, functions such as reliability and flow control have been left to higher layers. Hence this stack uses UDP datagrams. The link and physical layer services are used as provided, according to the connectivity available. To date, we have used 802.3 and SMDS, but others are possible.

By maintaining a standard interface between the data and signalling stacks and the underlying transport layer, we gain the advantage of complete portability among a large number of systems. However, there are two disadvantages. Firstly, this approach does not lead to the most efficient implementation. Secondly, it is difficult to incorporate any notion of priority in the system, either between the two stacks or among different connections in a single stack. Although neither of the link-layer protocols that have been used incorporate a model of priority, there are others that do. In these cases, it would be difficult to exploit this facility. This is discussed further in Section V.

2. *Services*

A brief description of the services provided at each layer of the protocol stacks is given here.

At the top layer of the signalling stack, the application layer, the following services are provided.

- init_NVCR(server_name). Before any other services are used, this is necessary to initiate a control connection to the server.
- end_NVCR. This closed the control connection.
- play(clip_name). To start playing a clip comprising video and/or audio.
- stop(clip_name). This service is terminal in that it ends the session for this particular clip, and releases all associated resources.
- pause(clip_name). To freeze the specific clip. All of the resources associated with the clip are retained.
- resume(clip_name). To continue a clip from the point at which it was paused.

To play or stop a clip, the application layer employs the services of the call layer. The following services are provided at this layer.

- create_call(channel[n]). The call comprises all of the channels required by the application, with all relevant details describing the channels, such as the quality of service of each component.
- close_call.

The protocols employed by the call layer to establish the end-to-end communication requirements of the application are dependent on the underlying network. For simple configurations, the call layer itself is sufficient to control this communication. In the general case, the call layer can make use of the connection layer to establish and control connections individually. This layer is based on the Internet ST protocol [11]. Currently, only the following services are provided by the connection layer.

- open_connection.
- close_connection.

Other services defined in ST include those related to the modification of a connection, either in terms of the number of targets or the quality of service. These are not used in the system described in this paper, as the application is point-to-point and none of the networks used provide quality-of-service guarantees.

The signalling protocols operate out-of-band from the data. Within a host, the interface between the two stacks lies between the application-level signalling and the real-time transport. At this interface, the application can invoke services in the data stack. For a transmitting host, the transport layer offers the following services.

- start_tx(channel_id, source_id, rate). To initiate transmission of data from the source specified, which is a file descriptor here, on the given channel, which is already established and again is specified as a file descriptor. The rate specification can be absolute, in terms of bytes/s, or can be a continuous function specified in a file. The former case is applicable to audio, where the data rate is constant, while the latter is applicable to video where the data rate is continuously variable during the clip.
- stop_tx(channel_id). To cease transmission and release the resources associated with this transmission at the transport layer.

- pause_tx(channel_id). To stop transmitting while retaining the channel resources.
- resume_tx(channel_id). To continue transmission from the point at which a pause was issued.

The complimentary receive services are provided by the transport layer on a receiving host. One major difference between the two is that the receiver has no direct control over the data rates so, for a loss-less service, the receiver transport must be able to receive the data at the rate that has been negotiated by the signalling protocols during the set-up of the session. It is a matter for the call-layer protocols to ensure that there are sufficient resources at each end to satisfy the quality of service requested for the clip.

3. Design

The protocols are designed as a set of asynchronously communicating finite state machines. Each protocol is specified as a data structure comprising an array of states, each state having an array of triples of the form (event, next_state, action). The data structures are interpreted by a state machine executor (SME). The SME is triggered by events that occur in the system. There are four sources of event in the system.

Commands from the user.
Information received from the network.
Events generated internally during state machine transitions.
Internally generated time-outs.

The system receives these events in different ways. For example, events resulting from user commands are received via an interface to the windows application while events from the network occur via packets received from the network interface. In all cases the event is translated into a standard format known as a signal. These signals are placed in a single FIFO queue, which is read by the SME.

This design has a regular structure which allows the protocols to be easily modified and extended. However, it does not lead to the most efficient implementation in terms of execution time. For this reason, the control path, which includes all of the signalling stack and the interface between this stack and the data stack, has been designed in the way described so far, but some of the indirection has been by-passed in the design of the real-time data path. Once a data channel has been established, there is a direct path of execution from the data source to the network interface on the sending host. Similarly, there is a direct path from the network interface to the video and audio subsystem on the receiving host. Experience with the working system has shown us that this decision was the correct one in order to achieve the required data throughput throughout the system.

For convenience, a menu containing a collection of useful operations has been incorporated into the system. This is usually dormant, but can be activated from the keyboard at any time. The facilities provided from the menu include the ability for an operator to examine and manipulate the signal queue, and to examine statistics that are collected during system operation.

```
make control connection;
while (active)
  if ((event from windows interface) or
      (event from network control interface) or
      (event from internal timers))
    translate event to signal;
    place signal on queue;
  if (signal on queue)
    remove signal;
    invoke SME with signal;
  for (each active i/p data channels)
    if (there is any data to be read)
      read the data from network channel;
      pass the data to VA subsystem;
  update clock;
  for (each active o/p data channels)
    if (it is time to send some data)
      read the data from source file;
      send the data to network channel;
  if (keyboard input)
    invoke menu;
```

Fig. 6. Structure of Protocol Software.

4. Implementation

The system is implemented as a single user process in C. The underlying network system is built from BSD sockets. The software has been written so that it can operate either as a client or a server, with minimal changes. The few changes required are controlled by compile-time options. By minimizing dependencies on system calls, the code is totally portable between either Unix or DOS based machines, assuming the presence of a sockets library for DOS.

An overview of the implementation is given by the pseudo-code in Fig. 6. On start-up, the system first attempts to make the control connection. If it is operating as a server, it waits indefinitely for the client to make contact. The client will do this in response to a user command from the windows application. Once the connection is established, the main control cycle is entered. Here, the system cycles a loop in which it alternately generates and services control events and then services the set of data paths that have been established. Thus events generated from the sources listed in Section IV.3 are detected and translated to signals on the signal queue. A signal may then be removed from the queue and used to invoke the SME. Following this, the data path is serviced. The set of active data input channels is checked for incoming data. Any data present is read and transferred to the video/audio subsystem. The data output channels are then serviced. The elapsed time since the last cycle is checked against the transmit period for each channel. If it is time to send data, it is read from the source file and sent across the network. The final action is to invoke the facilities menu if an operator has requested it from the keyboard.

It is important that only one event is serviced during each pass through the cycle. The asynchronous communication between the state machines ensures that it is not possible for a sequence of control actions to occupy the CPU contiguously at any time. In this way, the execution of the data path is given some priority, and the maximum possible data throughput sustained.

V. Experimental Results

To test the system, a clip of approximately 60 seconds duration, recorded from a film on a laser disk, was played back from the server onto the client. Measurements were taken on the client and the server, and also on a network analyzer attached to the network. A subjective assessment of the quality of the play-back was made through observation of the client.

The recording process allows us to vary the following parameters.

- Video compression factor.
- Audio compression factor.
- Video frame rate.
- Video picture size.
- Number of audio channels (mono/stereo).

Since the parameters related to the video component have a more significant impact on the data rates required, a number of recordings of the same clip were made with different values for these parameters but with the same audio settings of stereo with 4 bits/sample on each channel at a sampling rate of 9.45 KHz. The set of clips used are described by the parameters in Table I, which also shows the data rates associated with each clip. Each clip is identified by a name which is constructed as ⟨compression_factor: frame_size: frame_rate⟩, as defined in the table footnotes.

Detailed timing measurements were taken on the network and the client. The network measurements gave an accurate time stamp for each packet on the Ethernet, together with sufficient information to determine to which video frame a set of packets relate. The client measurements recorded the time at which each reassembled frame was transferred to the decompression subsystem. The quality of the clip was also assessed subjectively, with attention to frame loss or audio drop-outs, and observable synchronization between video and audio.

For local networking, tests were carried out using a single Ethernet segment. Initially, a private segment, on which no other traffic is present, was used. The restriction of using an unloaded network was then relaxed by performing the same test over a segment of the main Ethernet shared throughout the building. To study the effect of using a network that is not point-to-point, the final tests were carried out with the server attached to a T1 SMDS link, providing communication with the client through an SMDS-802.3 router, again using a shared Ethernet segment in the local area.

In the space available here, we refer to an examination of the performance of the system using a single clip, q2s2f20, from Table I. In summary, clip q2s2f20 could be successfully played back over each of the three network configurations without any data loss. Subjectively, the quality was judged to be as good as a play back from a local disk, with acceptable synchronization between the video and audio streams. Where the network configuration incorporated the shared Ethernet, this performance was dependent on the volume of other traffic on the segment during play back. Similarly, for the SMDS test, there would be a dependency on the other traffic through the router but for this test no other traffic was allowed as the data rates of the clip were very close to the capacity of the router.

TABLE I

Clip id	Mean frame size (bytes)	Mean video rate (Kbit/s)	Audio rate (Kbit/s)
q2[a]s2[b]f20[c]	4418	707	76
q2s2f25	4416	883	76
q2s3f20	5614	898	76
q1s2f20	6381	1021	76

[a] q(n) is the compression factor: for this clip, q2 = 0.86 bits/pixel and q1 = 1.25 bits/pixel.

[b] s2 = 256 × 160, s3 = 288 × 192

[c] f(n) = n frame/s.

TABLE II

Network	Audio Inter-packet time/s Nominal	Mean	Var.	Video Inter-packet time/s Nominal	Mean	Var.
Private LAN	0.088	0.092	3.2e-5	0.050	0.050	1.3e-5
Shared LAN	0.088	0.091	2.9e-5	0.050	0.050	1.3e-5
LAN/WAN	0.088	0.093	3.4e-4	0.050	0.051	4.4e-5

TABLE III

Network	Client Inter Frame Times/s Nominal	Mean	Variance
Private LAN	0.050	0.0496	5.3e-5
Shared LAN	0.050	0.0496	5.3e-5

Measurements on the network monitor showed that the packet dispersion on the network matched very closely the nominal rates at which the real-time transport was attempting to send data. In all cases there was some jitter about this nominal value, as summarized in Table II. The corresponding measurements taken on the client are summarized in Table III. These show a considerable increase in the delay jitter when compared to the network measurements.

For a 60 second clip, we ensured that this jitter had no degrading effect on the play back by incorporating a delay between the arrival of the first data packet on the client and the initiation of the decompression subsystem. This delay allowed the buffer occupancies to reach a level at which the subsequent jitter did not result in data being absent when required by the decompression hardware.

The detailed results of the tests are reported in Sections V.1–V.3 below. Again, we concentrate on an examination of the single clip q2s2f20.

1. The Private Point-to-Point Link

The dispersion of the audio and video packets for a single clip on the private Ethernet segment is shown in Figs. 7 and 8. These graphs show that the packets on each channel are sent according to the nominal rate control with some jitter accumulated between the sending transport process and the physical network. On both the video and audio channels, this jitter is characterized by mainly small, positive and negative, deviations about a mean value which is very close to the

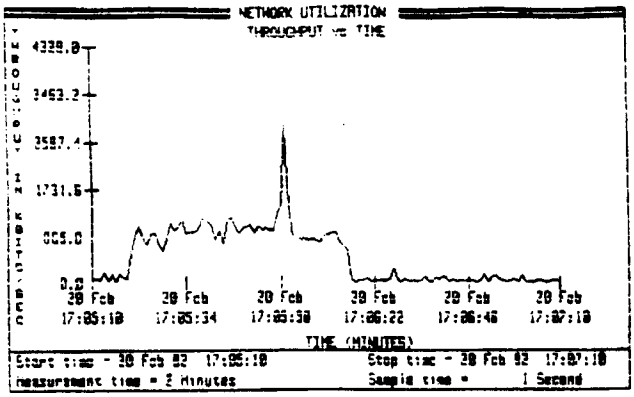


Fig. 13. Ethernet Trace for Unsuccessful Play Back.

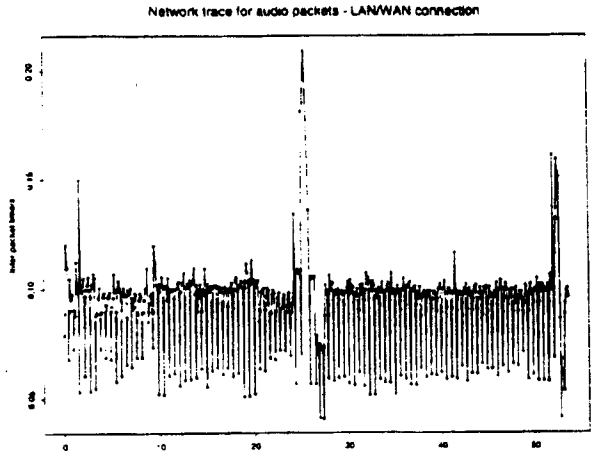


Fig. 14. Dispersion of Audio Packets over Ethernet/SMDS combination.

3. LAN/WAN Connectivity

Figs. 14 and 15 show the dispersion of the audio and video packets on the Ethernet segment during a successful play back across the Ethernet/SMDS combination. For this test, the Ethernet is the same shared segment employed in the previous test, and was used during a period of light load. The mean and variance of the inter-packet times on these graphs are given in Table II. The graphs clearly show increased jitter on both channels, compared to that accumulated on a point-to-point link. The inter-packet variance is an order of magnitude greater.

Although this test was successful, it is subject to the same restrictions as the shared LAN alone, and also to the load on the router. The presence of the router adds to the sensitivity of the play back as the maximum throughput of the router, an early experimental prototype, is approximately 1 Mbit/s, which is very close to the bandwidth required for the clip. Hence this test was run with no other traffic permitted through the router, and the results only show that it is possible to run the jukebox over the existing SMDS/Ethernet combination under a controlled scenario. The impact of the router on the packet dispersion is significant even in the absence of any other load on it. As before, for the 1 minute clip, the pre-buffering is sufficient to absorb this level of jitter.

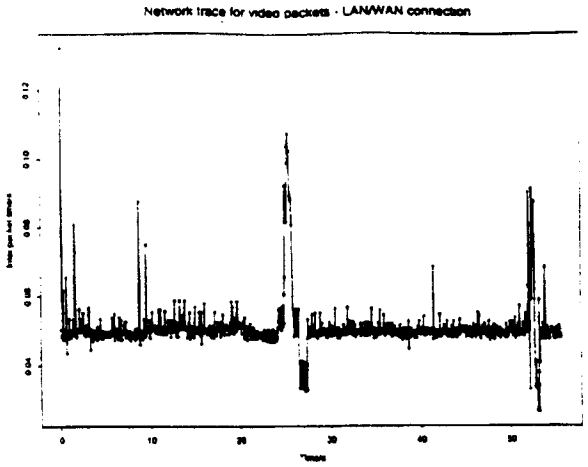


Fig. 15. Dispersion of Audio Packets over Ethernet/SMDS combination.

VI. Discussion

The results of the experiments that have been conducted show that applications with real-time requirements, such as the Video Jukebox, can be supported on standard currently available equipment. The Video Jukebox can be used in local stand-alone operation with a local disk, or as a networked service from a remote server which has the potential to support multiple channels and multiple clients concurrently. From a user's point of view, the change from a local to a networked service is seamless. In this section, we examine some of the implications of our results, and examine some of the limitations of existing systems.

1. Networking Multi-Media Applications

There is no doubt that the tremendous decrease in the cost of compression technology is leading to the wide introduction of video systems; forecasts predict an increase in the video conferencing equipment market from around $150 million today to anywhere from $240 million to $1 billion by 1995 [12]. There are many applications for which we consider our system to be adequate. Over the last two years there has been a rapid move towards using multimedia material in training, particularly where large numbers of people need to be trained rapidly. Current systems use either video tape or laser disc, with some moving towards compact disc based digital systems (either DVI [13] or CD-I). There are advantages and disadvantages with all these systems. Tape is not suited to random access; laserdisc and CD cannot be easily edited and, to be cost effective, need to be produced in large quantities. There is certainly a place for a video server for class room based teaching which stores random access, editable multimedia material. Organizations such as airlines need to train large numbers of cabin crew quickly on basic customer service principles and would benefit from a classroom in which multimedia material could be easily used.

One step removed from the system developed here are applications which require a simplex channel of live video. Again the delay requirements are not so stringent as for a

two way video conference and many people have proposed that such a service could have wide applicability. In particular remote support, maintenance, and monitoring of anything from power stations to aeroplanes is an interesting area. Airlines have a requirement for such a system to help reduce aircraft downtime caused by unscheduled maintenance at airports where they do not have a service depot.

There are three main factors that need to be addressed in extending our prototype into a robust and versatile system.

- Presence of other traffic, particularly over a WAN.
- Multiple clients using a single server.
- Extension to applications having more stringent real-time demands, such as two-way video communication.

These have consequences for the transport and network layers, as described below.

1.A. *Video and Audio Transport*

Real-time data transport is essential to support video and audio streams. One of our initial experiments with the Video Jukebox used a request-response model for data transport, based on TCP. In this system, the client would request data from the server at times initiated by the codec, and the server would respond by sending this data over a TCP connection. This approach was successful in very limited circumstances, but generally exhibited poor performance. By placing the source of timing in the server and using this to drive a rate-controlled transport layer, we can achieve more accurate timing and higher throughput.

For a system like the Video Jukebox, the existing protocols will be adequate for operation in many situations. Consideration of the factors listed above shows that there are two areas where we could usefully enhance the existing protocol. Firstly, the rate control should be extended to operate in a burst mode. It is not possible in practice to send data at a constant mean rate with total precision. Inaccuracies occur from the source timing itself, and are also introduced by other variable delays in the server. The jitter resulting from the disk access on the server for a single clip could be minimized, by contiguous file placement for example, but this would reappear if the server had multiple clients viewing different clips simultaneously. A file system that is designed to support real-time data streams, as described in [14] for example, will help but it is not possible to eliminate jitter throughout the entire system so the rate control should be extended so that variable delays can be accommodated. It is difficult for the transport to accommodate jitter within the network, which is addressed in the next section, but for the other sources, a variable transport data rate can absorb the jitter. The rate specification should include a nominal rate, corresponding to the mean data rate of the channel, with the addition of a higher rate that can be used periodically when the actual data rate on the channel has not matched this nominal rate. (Note also that a more adaptive rate control mechanism may also be appropriate to support data sources that are more bursty by nature. MPEG-compressed video could be an example of this type of source.)

Knowing when to switch data rates is the second area for enhancement. In a limited well-controlled environment, where the types of delay expected are known in advance, the switching period can be set a priori. This was in fact done on the video jukebox to achieve the same effect as the pre-buffering employed. This approach is fragile as it takes no account of the actual behavior of the channel on the network and in the client. A more satisfactory solution is to incorporate feedback from the client to the server giving the server knowledge of the actual data rates being achieved and the level of buffer occupancy. The transport can then adjust dynamically to maintain the real-time requirements of the channel. This type of facility is analogous to window-based flow control such as that employed in TCP, but with the timing under the direct control of the source, and feedback provided only when adjustments are required to this timing.

We also need to reconsider the requirements in order to support multi-media applications other than the video jukebox. Meeting the real-time requirements of the jukebox is eased by the fact that there is no interaction between client and server on the data paths. This gives us the scope to use pre-buffering on the client to overcome jitter on the data channels. Some applications, such as multi-media conferencing, are multi-way and interactive and would therefore not permit this technique to be used. Those applications will place much more stringent demands on the network layer to meet the real-time requirements, as discussed further below.

These extensions to the transport layer mean that the protocol incorporates many of the features contained in some of the proposals that have emerged for a new standard transport layer in future high-speed networks. Examples include XTP and VMTP. Some of these protocols also include support for multicast, which is a useful feature for applications requiring multipoint distribution. There is currently no consensus on the relative merits of these candidates, and little practical experience in using them on real networks, but it is likely that one or more of these protocols will form an integral component of future networked multimedia systems.

1.B. *Networks for Video and Audio*

A more adaptive transport protocol of the type described above may not be sufficient to maintain real-time channels in a general network environment, particularly where the channels have harder real-time demands than those for the jukebox. Jitter due to the network is not easy to predict, particularly over networks that are not point-to-point, and especially over large WANs. Reports on the load on the Internet, for example, show highly variable levels of congestion with corresponding variations in delay. In the general case where a user wishes to operate any time at any location, it highlights the requirement for support for real-time data at the network level in addition to higher levels. There are two ways of by-passing this problem.

- Exclude other traffic from the network. Dedicating a segment of the network to the real-time application is a short-term solution that will enable a limited number of users to operate.
- Use a network that has some support for synchronous channels. FDDI is one candidate. ISDN could be used for connection over the wider area.

The first approach may be an acceptable way of introducing multi-media applications such as the video jukebox to selected users, and for gaining exposure for the benefits of these systems. It has obvious limitations in terms of cost, convenience, and connectivity. The second approach provides a number of potential platforms, particularly in the wide area. It is currently unclear in which direction the public service providers will move, and it is certain that developments here will be influenced as much by political and economic factors as they will by technical considerations. See [15] for further insight into the options available here. Our interest in this paper is bringing real-time services to computer users now, so we focus on the use of the large existing base of conventional packet-mode networks. The Internet for example provides huge global connectivity, and is still expanding. Further, packet-mode operation has the attraction of economy and scalability, and is in some ways more appropriate for applications requiring multicast, which is likely to be an important facility in many multi-media applications. For applications displaying bursty traffic characteristics, isochronous channels are not necessarily appropriate anyway, and can lead to inefficient use of network resources. With extensions to the network layer, existing packet networks can be used to support real-time channels without the restrictions imposed during our tests. Enhancements are required in two areas.

- Resource reservation.
- Dynamic resource control.

The requirement for reservation is illustrated by the tests on the Ethernet/SMDS network. In this case, the bandwidth required by a clip is close to the total capacity of the router. It is necessary to negotiate with components such as the router to claim the resources required during a session, and to incorporate resource control mechanisms within these components that ensure that a quality-of-service that is negotiated is indeed fulfilled. A signalling protocol for this purpose is defined by the existing ST-II standard, and this has been incorporated into the protocol architecture of the video jukebox. However, there are currently few implementations in commercially available routers of the resource control required at the IP level.

The use of a shared Ethernet for local connectivity has some limitations. The non-deterministic access in Ethernet means that users will always be subject to some loss of data. Alternative technologies, such as Token Ring, provide more deterministic performance and therefore may be considered a more suitable base for this application. However, as already noted, the characteristics of the video traffic are not necessarily suited to this type of access. Further, some data loss is tolerable, and the nature of the MAC protocol in Ethernet is such that rather than losing a lot of data, it is more common that data is just delayed during periods of high load. With a more adaptive transport protocol, and perhaps with larger buffers on the client, it is likely that a fair of degree of tolerance to this delay can be built into the system.

We believe that the behavior of our system during unsuccessful playback on the shared Ethernet is not an inherent limitation of the Ethernet, but a result of some specific details of the implementation of the networking software in the client. A "lost" Ethernet frame, which results in a partially assembled UDP packet in the client, together with the implementation of the timing in the PC network software, can lead to a temporary deadlock of the type observed during unsuccessful play-back. The required behavior, for this application, in the presence of an unreliable link layer is for the client to discard data within a very short time (determined by the application) if it cannot be successfully received, so that subsequent video frames can be received. We would expect to observe no more than occasional loss of a video frame in this case. With the software available to us, we were unable to experiment with such changes. This is an example of where specific details of protocol implementation, targeted at conventional data transfer, are not always appropriate for exchange of real-time data such as video.

The future promises to deliver higher speed networks in both the local and wide area, based on cell technology such as ATM. The efficient multiplexing of traffic on these networks provides the potential for bandwidth on demand, and widens the scope for networked multimedia. (The articles in [16] give a good overview of future trends in multimedia communication.) These networks however introduce difficulties of their own, in that the end-to-end protocols have to handle periodic cell loss. Note also, that high-speed networks in general present a different performance characteristic in their latency/bandwidth parameter which fundamentally changes the way in which protocols need to operate. These problems are still at the point of research, and we do not discuss them further here. See [7] for an introduction.

RN-01979

2. *Video and Audio Subsystems*

The video and audio subsystems used for this work were prototypes originally developed in 1991 by Videologic to explore operation from a local disk. We have found that operation over a network introduces some different requirements for these subsystems. As an example, one such requirement is in the data transfer process to the cards. With the prototype system it was necessary to pass whole frames of video data to the compression card. It could not reassemble fragmented frames. This causes problems because the underlying data transport was UDP datagrams. The average frame size was around 4 KBytes which was transferred as a single datagram which after encapsulation by UDP and IP is roughly 3–4 times the maximum transfer unit (MTU) on Ethernet. As a result the frame is fragmented into a number of packets which are transmitted back to back on the network. These packets are then reassembled into the original datagram by the client's network subsystem before the whole datagram is read into a user buffer on the client.

In order to reduce this bursty transmission of Ethernet frames, which can lead to frame loss in the receiving host, it would be useful to fragment the original video frame into a number of datagrams. Each datagram would correspond to a single network packet and the individual packet transmission onto the network could then be controlled by the server. This

means that at the client at some point these individual packets must be reassembled into the original frames. The current interface between the host and the compression subsystem forces this to be done on the host before the data transfer to the compression cards. Given by the load on the host machine processor it would be useful to explore whether this reassembly could be better done on the compression subsystem control processor.

Another area which has not been explicitly addressed so far is the synchronization of source and sink clock rates and the synchronization of separate video and audio streams. The source and sink sample clock rate problem can be tackled by modifying the effective display time of a sample or frame to compensate for the differing sample rates [18]. One possible mechanism for doing this is to monitor the receive FIFO depth and use this to modify the playout time of buffered frames or samples, in a manner similar to a phase locked loop, or indeed a similar loop could be used to control the playout clock frequency itself. Alternatively end system clocks could be synchronized by the use of NTP (Network Time Protocol) [19], though there are obviously cases where this might be impractical, such as between separate organizations.

As noted earlier, the video and audio in this system are not linked by timestamps or other means, so both relative sample clock drift or data loss on either channel will cause the synchronization to slip. Many of these issues of terminal synchronization and stream synchronization are being considered as part of RTP, the real-time transport protocol, which is currently an Internet Draft document and the reader is referred there for further discussion [20].

A final point concerns the compression algorithms and media quality used in this system. As has been noted earlier the video was compressed using the JPEG algorithm which was originally developed for compression of full color still images [21]. The development of single chip implementations of this algorithm has enabled it to be used at rates up to 30 frames/second, and has led to its use as a video coding algorithm [22]. The disadvantage of this algorithm compared to MPEG or H.261 [23] is that it makes no use of the temporal redundancy of successive frames. This makes the algorithm considerably less efficient. Typical figures suggest that MPEG gains a factor of 3 in the overall compression of many image sequences for an equivalent image quality [4].

The video sequences used in our experiments were displayed as 1/4 screen VGA at 20 fps in full color. At the bit rate used the image quality was slightly worse than VHS video when viewed at typical computer screen distances. Whether this image quality is adequate depends very much on the application, but certainly for some applications mentioned, such as general purpose training, it would be sufficient. One area that needs further research is the ability to develop image quality metrics that can be related to applications requirements.

The audio quality could probably best be described as high end AM radio quality, though we did run the audio in stereo which gave the impression that it was closer to FM radio quality even though the sample rate used was only 9.45 kHz. Again it is difficult to relate this to applications requirements, but it was perfectly adequate to replay the film soundtrack.

3. Disk Technology

Disk technology is one area of concern if the video jukebox application, or related applications such as video mail, are to become viable in the near future. Parallel access to multiple stored video streams, or multiple random access to a single stream incurs significant overheads as the read heads track from one section of the disk to another. This greatly reduces the maximum possible rate that can be sustained off the disk, as was shown in Section III.

This problem of disk rates is not however exclusive to multimedia applications. Over the last decade the processing power of single chip CPUs has grown at 50–60% per year while dynamic disk performance has merely doubled. As a result there is considerable activity throughout the industry aimed at improving disk performance. Many companies are considering multiple disk array subsystems, or developing disks with multiple active read heads. There is also the possibility of low cost solid state FLASH disks appearing over the next few years. This implies it should be possible to develop relatively low cost video servers capable of delivering multiple streams during the mid to late 1990's.

VII. Conclusions

This work reported in this paper has shown that it is possible to build networked video and audio systems using standard equipment that is available off-the-shelf. As expected there are limitations in this approach. These are in the data rates and the number of users that can be supported, and the classes of application that can be provided. However, within these limits, video compression technology has given us the capability to provide some networked multimedia services now. The experiments conducted have highlighted the areas where further progress is required, either to support applications with real-time interaction or to provide higher data rates. We conclude by stating these areas among our general observations below.

- It is not necessary to use an isochronous network to carry networked video and audio. Conventional packet-based networks can be used to support many multimedia applications, althouh some work is required in the protocols used, and the precise implementation of these protocols. In some ways, these networks are better suited to the application, particularly when the data streams are bursty, as with compressed video for example.
- The system characteristics required to replay stored video and audio, particularly end-to-end delay and delay jitter, are much less stringent than those required for real-time full duplex video conferencing. The best opportunity in the near term for developing multimedia products integrated with conventional computer systems is with applications dependent on the replay of stored video, not those requiring real-time full duplex conferencing.
- There are many system design issues that arise when compressed isochronous data is brought into a conventional computer system. These problems become much more acute if the system is networked.

- In the future, we seek disk throughput as being a greater limitation than network bandwidth. Thus, it will be necessary to provide specialized disks, or disk arrays, for a server to support a large number of data channels simultaneously.
- It is a requirement to provide real-time transport for multimedia applications. However, the overhead of protocol processing is not significant. It is not necessary to provide custom hardware for this purpose. Applications that require multicast may prove more of a problem in this respect.
- The reduction in data rates achieved just through the use of JPEG hardware means that bandwidth requirements of a limited number of users of a real-time system such as the video jukebox can be met by existing networks. Of greater importance than the total bandwidth in the network is the mechanisms provided for resource allocation.

REFERENCES

[1] G. Clapp, "LAN Interconnection across SMDs," *IEEE Network*, vol. 5, no. 5, pp. 25–32, 1991.

[2] W. D. Sincoskie, "System Architecture for a Large Scale Video on Demand Service," *Computer Networks and ISDN Systems*, vol. 22, pp. 155–162, 1991.

[3] P. V. Rangan *et al.*, "Designing an On-Demand Multimedia Service," *IEEE Communications Magazine*, vol. 30, no. 7, pp. 56–64, 1992.

[4] *Communications of the ACM*, vol. 34, no. 4, 1991. Special issue on Digital Multimedia Systems.

[5] CCIR Recommendation 717, "*Tolerances for transmission time differences between the vision and sound components of a television signal*," Dusseldorf, 1990.

[6] R. Gussella, "*The analysis of diskless workstation traffic on an ethernet.*" UCB/CSD 87/379, Computer Science Division, University of California-Berkeley, November 1987.

[7] D. Boggs *et al.*, "Measured Capacity of an Ethernet: Myths and Reality," *Proceedings ACM SIGCOMM-88*, pp. 222–234, 1988.

[8] S. Minzer, "A Signalling Protocol for Complex Multimedia Services," *IEEE J. on Selected Areas in Commun.*, vol. 9, no. 9, pp. 1383–1394, 1991.

[9] L. Crutcher & G. Waters, "Connection Management for ATM Networks," *IEEE Network*, vol. 6, no. 6, pp. 42–55, November 1992.

[10] W. Doeringer et al., "A survey of light-weight transport protocols for high-speed networks," *IEEE Trans. on Commun.*, vol. 38, no. 11, pp. 2025–2039, 1990.

[11] C. Topolcic (ed.), "*Experimental Internet Stream Protocol, version 2* (ST-II)", RFC 1190, October 1990.

[12] J. Johnson, "Video Conferencing," *Data Communications*, Nov. 1991.

[13] G. D. Ripley, "DVI—A Digital Multimedia Technology," *Communications of the ACM*, vol. 32, no. 7, pp. 811–822, 1989.

[14] P. V. Rangan & H. M. Vin, "Designing file systems for digital video and audio," *Operating Systems Review*, vol. 25, no. 5, pp. 81–94, 1991.

[15] J. Sutherland & L. Litteral, "Residential video services," *IEEE Communications Magazine*, vol. 30, no. 7, pp. 36–41, 1992.

[16] *IEEE Communications Magazine*, vol. 30, no. 5, 1992. Special issue on Multimedia Communication.

[17] L. Kleinrock, "The latency/bandwidth tradeoff in gigabit networks," *IEEE Communications Magazine*, vol. 30, no. 4, pp. 36–40, 1992.

[18] W. Montgomery, "Techniques for packet voice synchronization," *IEEE J. on Selected Areas in Commun.*, vol. 1, no. 6, Dec. 1983.

[19] D. L. Mills, "Network Time Protocol (version 3)—specification, implementation and analysis," *Internet RFC 1305*, Mar. 1992.

[20] H. Schulzrinne, "A transport protocol for audio and video conferences and other multiparticipant real-time applications," *IETF Internet Draft Document*, Oct. 1992.

[21] ISO/IEC JTC1/SC2/WG10, "Digital compression and coding of continuous-tone still images," Draft 1991.

[22] B. Szabo & G. Wallace, "Design considerations for JPEG video and synchronized audio in a Unix workstation environment," *Usenix Summer 1991 Proceedings*, pp. 353–368.

[23] N. D. Kenyon & C. Nightingale, "AudioVisual telecommunications," *BT Telecommunication Series*, Chapman and Hall, 1992.

Laurence Crutcher (M'90) received the B.Sc (1985) and Ph.D (1989) degrees from the University of Bristol, UK. At the University of Bristol, he carried out research into computer architectures for the implementation of communication protocols, and lectured in the fields of computer architecture , real-time systems, and communication systems. In 1989, he joined Hewlett Packard Laboratories, where he carried out work in a number of areas in the Network Technology Group. This included work on protocols for high-speed networks, and protocols to carry real-time traffic over standard networks. In 1992, he joined the Center for Telecommunications Research at Columbia University, where he worked on network control and management for Gigabit networks. Since September 1993, he has been a software engineer at Market Vision, New York, NY. His research interests include multimedia systemsand network architecture and protocols.

John Grinham received the B.Sc (1984) degree from the University of Bristol, UK. Between 1984 and 1986 he worked on optical signal processing techniques at Cossor Electronics. In 1986, he joined Hewlett Packard Laboratories. Here he has carried out work in a number of areas within the Network Technology Group, and is currently working on networks and systems for multimedia applications.

Item -10-

The Ninja Jukebox

Ian Goldberg, Steven D. Gribble, David Wagner, and Eric A. Brewer
The University of California at Berkeley
{iang,gribble,daw,brewer}@cs.berkeley.edu

Abstract

We present the design and implementation of the "Ninja Jukebox", an infrastructural service that allows a community of users to build a distributed, collaborative music repository that delivers digital music to Internet clients, and that performs simple collaborative filtering based on users' song preferences inferred by the service. The Jukebox, implemented in Java, was designed to allow rapid service evolution and reconfiguration, simplicity in participation, and extensibility. We demonstrate that our careful use of a distributed component architecture enabled rapid prototyping of the service, and that our use of carefully designed, strongly typed interfaces enabled the smooth evolution of the service from a simple prototype to a more complex, mature system.

1 Motivation

The Internet is evolving towards a *service infrastructure:* a network of rich, robust, and often professionally maintained services that are conveniently accessible to people through the web. However, the fact that these services rely on the web to present their content effectively restricts their users to be human; the lack of structure and well-defined types in web content makes it all but impossible for computer programs to interact with most Internet services, despite the obvious benefits of being able to do so (such as service composition, richer search and information access services, etc.). Several recent efforts have attempted to introduce such structure and typing to the web, such as the WIDL [12] and WebL projects [16], or the ongoing W3C XML developments [6].

The UC Berkeley Ninja project[1] is pursuing a complementary path to these efforts: we are building an infrastructure for supporting fault tolerant, highly available, scalable services composed of a number of well-circumscribed components, each of which exports a strongly-typed, programming-language level interface [10] accessible using RPC-like mechanisms [4]. Explicitly exposing service interfaces and making use of strong typing has a number of benefits, including forcing authors to carefully design the boundary between their services and the rest of the world, making those services accessible to programs, and allowing the composition of services by infrastructural elements. We believe that when a large number of such services are deployed, a network-externality effect will occur, causing the power of an individual service to be greatly enhanced by interaction with the many other available services.

In this paper, we describe one such service: the *Ninja Jukebox.* The Jukebox allows a community of users to collaboratively build a distributed music repository out of both music CDs and MP3 files stored in local filesystems, and to use simple collaborative filtering to allow individual users to filter their music preferences according to other community members' explicit and implicit recommendations. In section 2, we discuss the design rationale that went into the Ninja Jukebox, and reflect on how the Ninja project's service philosophy influenced this design. Section 3 describes our Java-based Jukebox implementation and how we smoothly evolved it, and section 4 presents some of the limitations of our implementation and the lessons we learned while building it. Finally, in section 5, we present related work.

2 Design Philosophy

The Ninja Jukebox application was originally conceived of to "scratch the itch" of several graduate students: to be able to harness the large number of unused CD-ROM drives in the 100+ node Berkeley network of workstations (NOW) [3] and present a single, unified view of all music in all drives. Over time, the Jukebox has vastly evolved in complexity and richness. It now transparently supports both raw audio CDs in CD-ROM drives and MP3 files in local filesystems, and it performs authentica-

[1] Project home page: `http://ninja.cs.berkeley.edu`

tion and access control in order to adhere to copyright laws. It exports both a programmatic interface and an HTML interface for backwards compatibility with browsers; its programmatic interface includes a collaborative filtering service that deduces users' song preferences, and allows one to construct song playlists based on simple boolean combinations of other users' preferences.

The Ninja Jukebox was designed with several specific goals in mind. The first goal was that the Jukebox should be a communal, collaborative service. Individuals should be able to add or retract their personal collection of music from the Jukebox as they please, without requiring special intervention from a centralized administrator. This implies that contributors should be given as flexible as possible of a "service contract"—they must be allowed to retain control over their own contributions, while still ensuring that the overall Ninja Jukebox service maintains as stable as possible of a view to the rest of its users. The Jukebox service therefore must be able to adapt to changing group membership by gracefully masking unpredicted failures or disappearances.

Another goal was for the Jukebox service to retain flexibility, extensibility, and the facility for rapid evolution. As the evolution of the Jukebox has explicitly demonstrated, applications are not cast in stone, and services should not remain immutable once they have been released and are in use by applications and users. We therefore wanted our infrastructure to admit the evolution of its services, and we wanted to design the Jukebox service in such a way as to most easily allow it to be extended in unforeseen ways.

2.1 Design Implications

In order to meet the above goals, we made the following three explicit design decisions: the adoption of a distributed component architecture to decompose the Jukebox service into a small number of carefully chosen, functionally decoupled pieces, the imposition of a rich, strongly typed interface on these components (including carefully chosen data structures that precisely describe the contents of the Jukebox), and the use of soft state to achieve eventual consistency in the Jukebox.

Disciplined use of a distributed component architecture: as exemplified by Sun's Jini [21] and Corba [20], distributed component architectures advocate the use of an object-oriented language to decompose applications into smaller, self-contained objects, and the distribution of those objects across machine boundaries, relying on mechanisms such as RPC to perform inter-object communication. Component architectures make it simpler to begin with and maintain a clean design throughout the service's lifetime: the separation into objects allows for a separation of concerns, a tenet of good software engineering.

In the Ninja Jukebox, we decomposed our service into three major components, each respectively responsible for: (1) managing local collections of music (independent of physical and logical format), (2) the integration of many such collections of music and the maintenance of metadata about the music (such as users' song preferences), and (3) the client-side retrieval and playing of music from the service. This deliberate decomposition is what ultimately allowed the Jukebox to evolve so painlessly—each component's functionality is well encapsulated and isolated from other components, meaning that these components can internally evolve without affecting the rest of the system, and that new components can be added that compose with existing pieces to enhance the overall service. For example, the component responsible for managing local collections of music encapsulates information and access mechanisms particular to a music format, and thus the transition supporting only audio CDs in CD-ROM drives and also supporting MP3 files stored in a file system merely involved introducing a subclass of that component. Similarly, we could envision adding subsequent subclasses that would contain all music available from popular music web sites (such as `http://www.mp3.com`), or would serve as a gateway to receive music broadcasts (such as MBONE vat sessions).

Strongly-typed interfaces: in our opinion, the use of a distributed component architecture is only a partial step towards a properly decomposed service: the careful design of the interfaces between those components is a second, crucial step. An interface to a component is a declaration of both syntax and semantics, and as such is a contract that binds the component author to maintain those semantics even when the component is enhanced or extended through subclassing. Furthermore, the API to the service ultimately dictates the expressive power that clients of that service have available to them. We believe that an infrastructure service is defined by its interface and a declared set of guarantees about its performance and availability.

In the Jukebox, our APIs include data structures that richly describe content. These structures enable intelligent applications such as clients that group music on arbitrary terms, or that allow users

to construct playlists based on either explicit declarations or inferred preferences gleaned from the service's observation of their listening history. This focus on strongly-typed interfaces helps remove barriers to rapid service evolution by forcing service authors to carefully design and explicitly declare each of their components' interfaces, and therefore their implied service contracts.

Use of soft state to achieve eventual consistency: as a side-effect of making the Jukebox collaborative, we could not rely on any particular person's contributions to remain available. We thus designed the infrastructure so that a contributor periodically announces the presence of his/her music to a common master repository in order to add music to the overall Jukebox. The act of a person adding music to the Jukebox is therefore treated as a hint rather than a promise: components cannot rely on that music being there, and they must gracefully handle the case in which a particular song abruptly becomes unavailable. We also treat entries in the master repository as a lease, and expire them if the periodic announcements stop. The master repository correspondingly contains an approximate view of all available music: this view continually approaches the correct view over time. This leased approach is also used in our authentication mechanisms: when a client requests a song from the Jukebox, it must first authenticate itself, the result of which is a capability that is good for a single use or for thirty seconds: the Jukebox components lazily time out these capabilities as necessary.

Not all state in the Jukebox is soft-state: users' song preferences, for example, are stored as hard state by dedicated, highly-available infrastrucure in what we call a "base"[10]. A base is composed of everything needed to build an available compute cluster, including system administration, a secure machine room, redundant networks, UPS, etc., and as such is an ideal environment in which to protect hard state.

3 Implementation

This section of the paper describes the implementation of the Ninja Jukebox service and client, and their evolution through three stages of functionality. The first version of the service only supported the playback of raw audio CDs from the CD-ROM drives of Jukebox servers. In the second version of the service, we added the ability to convert raw CDs into compressed MP3 files, and for those MP3 files to be played over the network; this second version also included authentication and access control mechanisms to enforce copyright protection. Finally, we added a simple collaborative filtering mechanism to the third and current version of the Jukebox.

We chose to implement the Ninja Jukebox service in Java, both because Java trivially enables distributed components and because the Ninja project has developed a significant amount of infrastructure in Java. This infrastructure includes authenticated remote method invocation (RMI) and a cluster-based service platform called "MultiSpace" [10] that was designed to support scalable and rapidly evolvable infrastructure services.

3.1 Ninja Jukebox v1.0: raw audio CD playback

As shown in Figure 1, the first version of the Ninja Jukebox implementation was decomposed into the following elements:

The SoundSmith: SoundSmiths are responsible for indexing and maintaining a structured collection of music. The version 1.0 SoundSmith indexes music on an audio CD in a local CD-ROM drive, making use of a service that acts as an HTTP to RMI gateway to provide programmatic access to an online "CDDB" database[15]. This database provides a mapping from a CD's track timing information to detailed information about the CD's author, song titles, and song durations. SoundSmiths periodically send beacons to the MusicDirectory; through these beacons, they both announce their existence to the MusicDirectory and present the list of songs that they maintain. Anyone that wishes to contribute music to the Jukebox must only run a SoundSmith that can index and serve that music. SoundSmiths can be started up and torn down at any time, as each SoundSmith is completely autonomous, and the beacons emitted by the SoundSmith are treated as soft-state by the MusicDirectory. A SoundSmith serves music by streaming it off of an audio CD from the CD-ROM drive of an infrastructure workstation and transmitting it in uncompressed .au format through an (untyped) HTTP interface.

The MusicDirectory: As previously mentioned, SoundSmiths periodically beacon their existence and a list of their music to the MusicDirectory. The role of the MusicDirectory is to keep track of these beacons, and to build up an integrated list of all available music and of all running SoundSmiths. Clients use the MusicDirectory as a level of indirection that shields them from needing to inde-

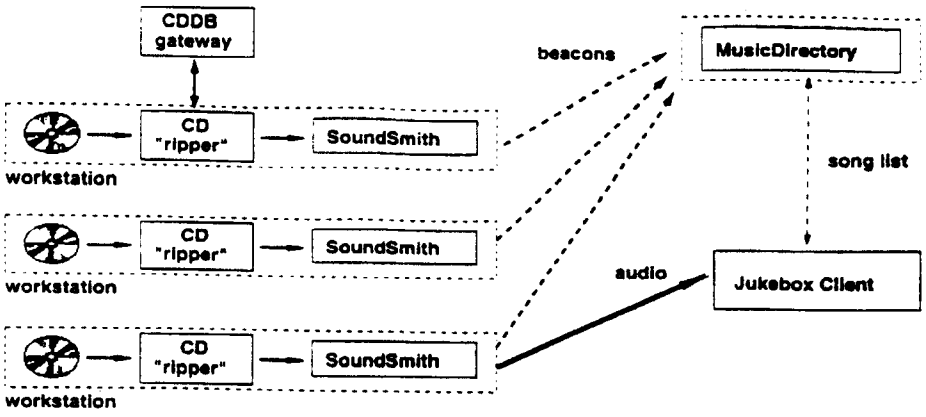


Figure 1: The Ninja Jukebox v1.0 architecture

pendently discover the location of all SoundSmiths in the Jukebox. Ultimately, this centralized MusicDirectory limits the scale of a Jukebox, since all SoundSmiths repeatedly send it listings of music.

Jukebox Clients: Jukebox Clients interact with a MusicDirectory to gather a listing of available music, and with many SoundSmiths to receive and play specific songs. We have currently implemented two clients. The first presents a graphical user interface to the user (figure 2), and allows users to build playlists of available songs. Music streamed to this client is shuttled to external music players that understand many music formats and have the ability to play music as it is streamed over the network. Internally, this client is decomposed into a GUI front end and a song selection back end. The GUI front end provides the user with controls for constructing playlists, and with familiar *play*, *stop*, *pause*, *fast-forward*, and *reverse* buttons. The song selection back end selects specific songs to play given the list of currently available music from the MusicDirectory, the user's manually constructed playlist, and events that are generated when the buttons such as *play* or *stop* are pressed. The second client is a proxy that converts between the APIs and data structures exported by the Ninja Jukebox service and HTML forms. This proxy allows conventional HTML browsers to access the Jukebox: music is streamed through the proxy to the browser, or presumably to the browser's helper applications that can actually understand specific audio formats.

This first version of the Jukebox service was well received even though it suffered from a number of drawbacks. The fact that all audio was transmitted in an uncompressed format resulted in excessive traffic on our local networks, greatly limiting

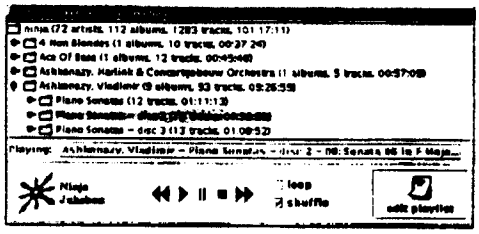


Figure 2: The Ninja Jukebox client GUI

the number of clients that could simultaneously access the Jukebox. Furthermore, the fact that music could only be served from audio CDs physically present in CD-ROM drives limited the amount of music that could be present in the Jukebox at any given time, since we had a limited (although large) number of CD-ROM drives at our disposal. Finally, the lack of any security infrastructure prevented us from widely releasing the Jukebox service and client, even within our department, since it would become trivial for users to violate copyright protection legistlation, either accidentally or deliberately.

3.2 Ninja Jukebox v2.0: MP3 playback and security

The separation of the Jukebox into the previously described components satisfied our primary design goal: to construct a collaborative service, in which anyone can contribute their collection of music to the Jukebox. A second design goal was to allow for the evolution of the service; in order to test this goal (and to satiate the demands of the clients of the v1.0 Jukebox), we extended the Jukebox functionality to

produce the v2.0 version of the service. This version of the service attempted to overcome the drawbacks of the v1.0 prototype by including two new major features: the transparent support of MP3 files, and support for access control and client authentication.

We also slightly modified the Jukebox by having SoundSmiths only report their existence to the MusicDirectory rather than the complete list of music that they manage; in the v2.0 infrastructure, clients discover SoundSmiths through the MusicDirectory, but then ask each individual SoundSmith for its list of locally available music. This modification drastically reduced the size of the SoundSmith's beacons, which eased the scaling bottleneck caused by the centralized MusicDirectory. This bottleneck became increasingly evident as the body of music stored in the Jukebox grew to over 4,400 songs (375 albums, accounting for more than 25 gigabytes of hard drive space and 320 hours of music).

3.2.1 MP3 Support

MP3 support was surprisingly easy to add to the Jukebox service. To do it, we simply created a subclass of the SoundSmith component that understood how to index and stream MP3 files on a regular filesystem instead of audio tracks from an audio CD in a CD-ROM drive. The data structures embedded in the SoundSmith's beacons are only metadata, and as such are totally independent of the specific format in which the music is actually kept. In order to play music, Jukebox clients interact with the MusicDirectory service to fetch an HTTP URL for a song; this URL is served by the SoundSmith that maintains the song. Because the song data is streamed to external music player software that happens to understand MP3 formatted music, the Jukebox clients never need to understand anything about the music format. When we deployed several of these MP3-aware SoundSmiths in our infrastructure, Jukebox clients suddenly became aware of a much larger set of available music, and transparently began accessing the newly available MP3 files.

The MP3 files maintained by SoundSmiths are created by helper daemons that batch convert music CDs to MP3 formatted files by first "ripping" raw audio from the CD, and then compressing that raw audio into an MP3 file and its associated artist and album metadata (figure 3). These daemons run in the background on all of our Jukebox workstations, effectively crawling the Jukebox for new music to MP3 compress and add to the Jukebox. While this conversion is happening, an audio CD SoundSmith can serve the music directly off of the audio CD;

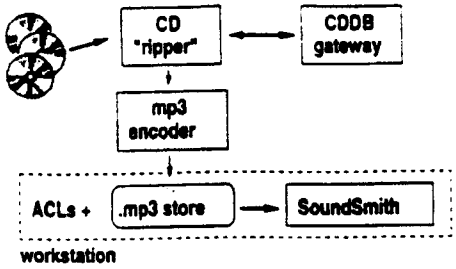


Figure 3: MP3 Support in the Jukebox v2.0

after the conversion is finished, the music can be served in the preferable MP3 format.

We attribute the ease with which we added support for MP3 files to the Jukebox infrastructure to our use of a distributed object infrastructure and to the strongly-typed interfaces between our Jukebox components. The ability to subclass in order to specialize the SoundSmith allowed us to maintain its RMI interface, and thus upgrade its functionality in a manner that was transparent to the rest of the Jukebox. Transparency was meaningful because of the presence of explicit interfaces between the components; achieving transparency in this case was a manner of maintaining both the syntax and declared semantics of the interface.

3.2.2 Security Infrastructure

For the Jukebox, the only relevant security issues are access control and authentication. Our authentication mechanism is based on SecureRMI, a variant of RMI—Java's standard remote method invocation protocol [17]—that we have developed to operate over a cryptographically-secured channel. With this tool in place, the access control problem becomes relatively easy: for each song in a SoundSmith, that SoundSmith maintains an ACL (a list of SecureRMI principals allowed to play that song). The access control mechanism thus is as simple as having the SoundSmith look up an entry in a list. The SoundSmith also hands out capabilities to authenticated principals that allow them to access specific songs for a limited amount of time: these capabilities are good for a single access, and expire if not used within 30 seconds. Note that the MusicDirectory does not need to authenticate the identity of clients, as it is entrusted only with a list of available songs and SoundSmiths, and not the songs' content. For the proxied HTML-based client to work, however, the proxy itself must be entrusted with its users' credentials, since HTML browsers do not

have the ability to interact with our SecureRMI infrastructure directly.

Currently, our policy for access control is relatively simplistic: a principal can only listen to a copyright-protected song if she has previously demonstrated knowledge of the song contents (e.g. by uploading it to the Jukebox); unrestricted access is given to music marked as non-copyrighted. This approach is inspired by legal considerations: if people can't abuse the Jukebox to gain access to music they don't already have, it seems unlikely that the Jukebox will be accused of violating copyright laws. However, the Jukebox could also accommodate more sophisticated policies for access control, such as support for group ownership where only one group member can listen to a song at a time, or a pay-per-use scenario under which royalties could be collected and submitted to the copyright holders. The flexibility of our design makes such variations on authentication quite straightforward.

Returning to the authentication mechanism, SecureRMI was the piece of the security architecture that demanded the vast majority of our security engineering effort. SecureRMI (optionally) authenticates the endpoints and then encrypts the remainder of the communication with a Triple-DES session key derived from a Diffie-Hellman key exchange. We also provide a certification infrastructure for endpoint public keys and tools for managing them; certificates bind the service's fully-qualified class name (or the client's identity) to the server's (or client's) public key.

Of course, there is nothing new about the concept of establishing a secure channel with the use of encryption [18, 22]. However, we feel that our implementation may be of interest primarily because it exists: we are not aware of any other free, Java implementation with similar functionality.[2]

One novel feature of our SecureRMI is that it provides transparent support for a very broad range of "authentication" technologies. We have abstracted away many of the irrelevant details of the algorithms to build a very general model of authentication. For instance, public-key authentication is implemented in `DSAAuthenticator` and `DSAVerifier`, which are subclasses of the generic `Authenticator` and `Verifier` classes; SecureRMI only references the generic `Authenticator/Verifier` superclasses, so it is ignorant of the details of their implementation. This architecture is very flexible: after the core infrastructure was in place, we later added a symmetric-key challenge-response protocol with about two days of coding.

As a result, extending the Jukebox to support pay-per-use access will require only minimal effort. We would just add a `PayPerUseAuthenticator` that, instead of sending a public-key signature for authentication, sends a digital coin. This is a direct result of our design goal that services be easy to evolve and extend.

Our general model of authentication also allows each collaborator to specify her own access-control policies for the music she serves; one SoundSmith could be serving music on an ACL basis, another could be serving only free music, but only to hosts in a certain domain, and others could be charging various amounts to listen to the audio stream. The flexibility provided by this mechanism further enhances the communal, collaborative nature of the Jukebox, by removing access-control policy from any central authority.

3.3 Ninja Jukebox v3.0: the collaborative DJ service

Most recently, we have extended the Jukebox service to provide song selection based on inferred user preferences as well as some simple collaborative filtering functionality. In the v1.0 and v2.0 Jukebox services, song playlists are manually constructed by users and successive songs to be played are chosen from these playlists by simple random selection. Our collaborative filtering extension refines this selection with an infrastructural "DJ" service that exploits individual and collaborative song preferences.

A key observation is that an infrastructure service may, over time, learn user song preferences by observing UI events. Songs that the user always "fast-forwards" past are probably songs the user doesn't like; in contrast, listening to a song until its completion may be an indication that the user enjoyed the song. This observation forms the basis for our preference inference mechanism. As we described in section 3.1, our graphical Jukebox client is decomposed into a GUI front end and a song selection back end. In our v3.0 Jukebox infrastructure, we have decoupled this song selection from the client executable, moving it instead into the network as a infrastructure service so that our song selection algorithms may be upgraded and evolved transparently without modifying code on the client side. This enabled us to extend the original unintelligent song selection algorithm by interposing on the selection interface.

In our DJ implementation, a rating storage ser-

[2] JDK 1.2 includes hooks so you can encrypt RMI communications with SSL, if you have a SSL library; but we do not know of any free SSL implementations for Java [8].

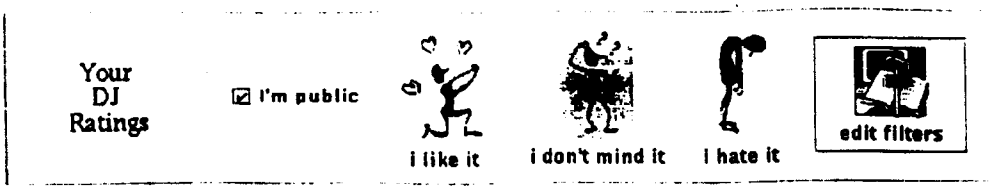


Figure 4: The DJ collaborative filtering client GUI element

vice in the infrastructure subscribes to client UI events; every time a user presses a button such as "fast-forward", a SecureRMI call is made into this DJ service to report the event. The DJ interprets these events as implicit hints about the user's song preferences, and updates a persistent database[3] on disk to reflect the new information about the user. Our prototype also allows users to explicitly specify their preferences about individual songs, if they like. Still, the advantage of transparent preference inference is that it requires no extra action on the part of the user.

A second key observation is that, when preferences for many users are all stored together in the infrastructure, there is a great opportunity to mine this data for cross-user information and to provide collaborative services [19]. We have implemented a simple collaborative filtering application for the DJ. By default, a user's preferences are regarded as private and are stored securely in the infrastructure, with no access allowed to third parties; however, we allow users to publicly export read-only access to their preferences to other users. Marking one's preferences as public allows one to share preferences between multiple users. For example, our implementation allows a user to temporarily use someone else's preferences for song selection (assuming, of course, that those preferences have been explicitly marked as public). More interestingly, a user may combine the preferences of multiple other public users and use the result to drive the Jukebox client's song selection algorithm. This is a useful way to accomodate multiple listeners with different preferences; for example, in an shared environment in which several students occupy the same office, a useful combination would be to play songs that are in the intersection of the students' sets of likable songs.

The DJ extensions to the core Jukebox service resulted in minimal changes to the existing codebase; rather, the extensions were mostly encapsulated within the new DJ component that was added to the Jukebox infrastructure. The required changes to the existing codebase were limited to modifications to the Jukebox client's song selection algorithm to request a playlist from the DJ service, and to the enhancement of the Jukebox client front end to send a copy of all relevant events to the appropriate rating storage service. We also augmented the Jukebox client GUI to include controls that allow the user to explicitly indicate preferences for specific songs (figure 4).

[3] We actually used a distributed, persistent hash table to keep track of user preferences. This hash.table (described in [9]) is partitioned and replicated across nodes in a dedicated workstation cluster, and provides the DJ fault-tolerant access to the persistent user preferences data.

4 Discussion

In this section of the paper, we first present several lessons that we learned about using Java as a service construction language, and then we discuss several limitations of the current Jukebox implementation.

4.1 Java as a Service Construction Language

We were surprised to find that the decision to use Java as a rapid prototyping tool met with mixed results. Certainly, Java's high-level programming model made for extremely rapid prototyping: the first version of the Jukebox service was built in 2 days by a team of 3 students. Java's strong typing also encouraged modularity, which made it easier to extend and evolve the service several times: once to migrate from playing CDs in real-time towards serving as a shared MP3 repository, later to extend the service to add a security model, and a third time to transparently learn song preferences and to add support for collaborative filtering. In all three cases, strong typing helped assure the separation between client code (which should change rarely) and networked services (which may evolve frequently) that was a key ingredient to success. Also, the tight coupling of RMI with Java, and the existence of the

Ninja SecureRMI infrastructure made distributed programming less painful.

What we didn't anticipate is that there were some negative aspects to using Java and RMI. When you change the implementation of some relevant class on one RMI endpoint, to avoid class checksum errors you must grab the new source code and recompile on all other endpoints too. Thus, updates to the service code require synchronized updates at all RMI endpoints, which is an administrative annoyance. Moreover, though we didn't realize it at first, if we had used RMI for all of our external service interfaces, the situation would have been far worse: each upgrade to the Jukebox service would potentially have required the clients to be updated too, a terrible scenario for service evolution! Fortunately, we got lucky: most of our external interfaces that changed used HTTP, not RMI. Our interpretation (in retrospect) is that we *should* have done a better job of picking strongly-typed interfaces to the outside world (rather than having any untyped HTTP connections) and frozen these interfaces from the outset, but we didn't. We gained considerable leverage from the use of narrow, strongly-typed interfaces between internal Jukebox components, and if we were to re-implement the Ninja Jukebox, we would strive to do the same for all of our external interfaces as well.

4.2 Limitations

Our current prototype of the Jukebox service has a number of limitations. First, the Jukebox is not intended—in its current incarnation—as a wide-area distributed service. Instead, we have focused on providing service within a single organization. As an example, the MusicDirectory service is currently centralized, which means that it would quickly become a bottleneck if we moved to a wide-area usage scenario. We have also made the simplifying assumption that all nodes in the system are relatively close to each other (in terms of network latency and bandwidth), so that from the client's point of view all SoundSmiths are created equal.

Although a wide-area Jukebox service would be limited by the capacity of the underlying network, with some more work we could extend Jukebox to address wide-area concerns. Two changes would be required: (1) the MusicDirectory service would have to become wide-area aware, using standard techniques such as replication, caching, and aggregation to distribute song listings around the world, and (2) we might want to replicate MP3's across the wide-area, using pre-fetching and caching to reduce the load on the network. Neither of these changes are conceptually difficult; we built the Jukebox because it was a service we wanted, and so we ignored these aspects.

A second important limitation is that our current implementation is very naive about multimedia operations. The MP3 data is transmitted over a HTTP connection, and thus inherits all of the problems of TCP for multimedia data: no quality-of-service guarantees, potentially high latency, unwanted buffering, and so on. There is also no rate limiting; we merely blast as fast as we can, which runs the risk of overloading the network. Nor are our clients particularly sophisticated about multimedia issues: our MP3 player doesn't do real-time scheduling, so during heavy paging we occasionally experience playback glitches. Nonetheless, these issues are largely orthogonal to our research; instead, we focused on testing the hypothesis that we can rapidly build a highly evolvable service if we carefully use component architectures and strongly typed interfaces, and thanks to this extensibility we believe a future version of the Ninja Jukebox could easily include better multimedia delivery technology.

Thirdly, our current prototype has poor performance for the Java security operations. Right now, we are using a pure-Java cryptographic library, with no JIT, and as a result the public-key operations are very CPU-intensive: the initial SecureRMI handshake currently takes about 4 seconds to complete. Of course, these numbers could be dramatically reduced by any of many techniques (native code, precomputation, caching, session-reuse, etc.), but so far the performance impact has been relatively innocuous.

5 Related Work

Keeping collections of audio files in a net-accessible way is obviously not a new idea. The simplest way to publish one's music collection to the net is just to make it accessible as a WWW or FTP archive. Many people do this today, but the utility of unconnected collections of audio is low. In order to make these collections more useful, dedicated MP3 search engines such as `mp3.lycos.com` have appeared. These search engines try to be your "one-stop shopping" for MP3's, by telling you where on the Internet you can find your favorite pirated songs.[4] More recently, commercial jukebox prod-

[4] Lycos itself, of course, does not illegally publish copyrighted material.

ucts have become available that allow you to organize and play locally-stored MP3's, but these products typically do not permit sharing between users, nor do they offer collaborative or interactive features.

This simplest kind of jukebox system is missing a number of benefits that the Ninja Jukebox provides. Simple directories of MP3 files offer no cohesive framework for security-related features such as authenticated or pay-per-use access. In addition, our component architecture allows the SoundSmiths to be active and easily updatable participants in the transmission of the audio, as opposed to merely serving a static file. This allows features such as transparent format conversion (`.wav` files on file systems, raw audio on CD-ROM drives, or MP3 files on file systems) and support for multiple transport mechanisms (streamed audio over HTTP, or VAT audio over a multicast IP channel).

Another approach has recently come from SHOUTcast [11]. SHOUTcast is an "Internet radio" system that allows a site to serve an audio stream that can be picked up by multiple clients. The clients have to listen to what the SHOUTcast servers decide to play; they have no way to interact with the servers. Although each SHOUTcast server offers the same real-time audio stream to each of its clients, and though its name would imply something more clever, its underlying technology is just multiple simultaneous unicasts of the same data. SHOUTcast servers communicate with one or more central databases in order to register the names of the programs they are currently "broadcasting". These databases can be queried by client programs (like MP3Spy [13]) to allow users to choose what channels they would like to hear.

The largest difference between SHOUTcast and our work is that our goal was to provide a communal, collaborative, interactive jukebox, as opposed to a passive Internet radio station. That having been said, however, it would be possible for a SoundSmith to transmit any particular song over a true multicast channel [7]. SHOUTcast servers also do no user authentication; one might indeed imagine that an Internet radio service would have no need for such a thing. However, given the broad view of "authentication" taken by the Ninja Jukebox, one could see that implementing, for example, subscription-based access or pay-per-use access, could add value (better quality of service, for example) even to a non-interactive service like Internet radio.

A related approach is the Interactive Multimedia Jukebox [1, 2], a system that allows one to add a measure of interactive preference feedback to traditional broadcast paradigms.

More recently, the SDMI project is starting to tackle the issues associated with copyright control and rights management, using a combination of tamperproof hardware (or software!) on the client end as well as watermarking and other technologies [14]. We view SDMI as largely orthogonal to our work: we have focused on building a music delivery service, rather than on what is done after the music has been delivered.

There have been a number of projects involved in the delivery of audio and/or video over digital networks (for example, [5]); these projects mainly concern themselves with the technology of media delivery. In contrast, we have left that issue largely unaddressed, as it is orthogonal to our own goals; we were more interested in the mechanisms of the service, rather than the mechanisms of serving.

6 Conclusions

In this paper, we demonstrated that Java is a convenient language for the construction of infrastructural services, although there are several pitfalls and hurdles (such as performance, vagaries about the internals of its RMI facilities, etc.) that need to be addressed or avoided in order to successfully build such services. We also partially validated our hypothesis that infrastructural services which explicitly expose a strongly typed, programmatic API (as opposed to an unstructured interface designed only for humans) are conducive to the construction of complicated applications. Finally, we demonstrated that a distributed component architecture enabled the rapid development of an infrastructural Jukebox service, and that through the careful decomposition of the service into components and deliberate attention given to the design of the service's internal and external interfaces, we were able to smoothly evolve the first generation Jukebox into a more rich and mature service.

References

[1] K. Almeroth and M. Ammar. The Interactive Multimedia Jukebox (IMJ): A New Paradigm for the On-Demand Delivery of Audio/Video. In *Seventh International World Wide Web Conference (WWW-7)*. World Wide Web Consortium, April 1998.

[2] K. Almeroth and M. Ammar. An Alternative Paradigm for Scalable On-Demand Applications:

Evaluating and Deploying the Interactive Multimedia Jukebox. *IEEE Transactions on Knowledge and Data Engineering Special Issue on Web Technologies*, July/August 1999.

[3] Thomas E. Anderson, David E. Culler, and David Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 12(1):54–64, February 1995.

[4] Andrew D. Birrell and Bruce Jay Nelson. Implementing Remote Procedure Call. *ACM Transactions on Computing Systems*, 2(1):39–59, February 1984.

[5] William J. Bolosky, Joseph S. Barrera III, Richard P. Draves, Robert P. Fitzgerald, Garth A. Gibson, Michael B. Jones, Steven P. Levi, Nathan P. Myhrvold, and Richard F. Rashid. The Tiger Video Fileserver. Technical Report MSR-TR-96-09, Microsoft Research, Advanced Technology Division, April 1996.

[6] The World Wide Web Consortium. Extensible Markup Language (XML) version 1.0. `http://www.w3.org/XML`, Feb 1998.

[7] Stephen E. Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. An Architecture for Wide-Area Multicast Routing. In *Proceedings of SIGCOMM '94*, University College London, London, U.K., September 1994.

[8] Li Gong. New Security Architectural Directions for Java. In *Proceedings of IEEE COMPCON*. IEEE, February 1997.

[9] Steven D. Gribble. Simplifying Cluster-Based Internet Service Construction with Scalable Distributed Data Structures. Ph.D. Qualifying exam, available at `http://www.cs.berkeley.edu/~gribble/papers/quals/sdds-cluster.ps.gz`, April 1999.

[10] Steven D. Gribble, Matt Welsh, Eric A. Brewer, and David Culler. The MultiSpace: an Evolutionary Platform for Infrastructural Services. In *Proceedings of the 1999 Usenix Annual Technical Conference*, Monterey, California, USA, Jun 1999.

[11] Nullsoft Inc. SHOUTcast Service home page. `http://www.shoutcast.com/`.

[12] WebMethods Inc. WIDL—Web Interface Description Language. *World Wide Web Journal*, 1997. Special issue—XML: Principles, Tools, and Techniques.

[13] Game Spy Industries. The MP3Spy Client home page. `http://www.mp3spy.com/`.

[14] Secure Digital Music Initiative. SDMI Portable Device Specification, part 1, version 1.0, July 1999. `http://www.sdmi.org/`.

[15] Ti Kan and Steve Scherf. CDDB Specificaton. `http://www.cddb.com/ftp/cddb-docs/cddb_howto.gz`.

[16] Thomas Kistlera and Hannes Marais. WebL—A Programming Language for the Web. In *Computer Networks and ISDN Systems (Proceedings of the WWW7 Conference)*, volume 30, pages 259–270. Brisbane, Australia, April 1998.

[17] Sun Microsystems. Java Remote Method Invocation—Distributed Computing for Java. `http://java.sun.com/`.

[18] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM*, 21(12):993–999, December 1978.

[19] Firefly Network. Firefly Passport. `http://www.firefly.net`, 1995.

[20] The Object Management Group (OMG). The Common Object Request Broker: Architecture and Specification, February 1998. `http://www.omg.org/library/c2indx.html`.

[21] Jim Waldo. Jini Architecture Overview. Available at `http://java.sun.com/products/jini/whitepapers`.

[22] Edward Wobber, Martin Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos Operating System. *ACM Transactions on Computer Systems*, 12(1):3–32, Feb 1994.

RN-01966

Item -11-

Multimedia Federated Databases on Intranets : Web-Enabling IRO-DB

Georges Gardarin

Laboratoire PRiSM URA CNRS 1525
Université de Versailles-St Quentin
45 avenue des Etats-Unis
78035 Versailles Cedex - FRANCE
email : (1) <firstname>.<lastname>@prism.uvsq.fr

Abstract. *Integrating semantically heterogeneous databases requires rich data models to homogenize disparate distributed entities with relationships and to access them through consistent views using high level query languages. In this paper, we first survey the IRO-DB system, which federates object and relational databases around the ODMG data model. Then, we point out some technical issues to extend IRO-DB to support multimedia databases on the Web. We propose to make it evolve towards a three-tiered architecture including local data sources with adapters to export objects, a mediator to integrate the various data sources, and an interactive user interface supported by a Web browser. We show through an example that new heuristics and strategies for distributed query processing and optimization have to be incorporated.*

Key words. *Interoperable database. Federated database. Object-oriented database. Remote data access. Schema integration. Multimedia. Web. Query processing.*

1. Introduction

Object-oriented multidatabase systems (also referred to as federated databases or heterogeneous databases) represent the confluence of various trends in computer science and technology [1], among them object-orientation [2], distributed databases, and interoperability. Recently, the Internet has become the major vehicle in networking industry for information access and dissemination. The Web as a service on top of the Internet or Intranets focuses on transparent navigation and hypermedia document oriented information access. Thus, today there is a need to integrate the object-oriented multidatabase technology within multimedia Web-based systems, both for Intranet applications and Internet services. This paper discusses the integration of multimedia and Web techniques within the IRO-DB federated database system.

While much of the early work in federated databases concentrated on relational technology [3], some projects have been developed at the beginning of the 90's based on object models. Pegasus [4] was one of the first developed at HP Lab. It is centered around a global object model to which local objects (e.g., tables from a relational DB) are mapped. The global model is close to the object-relational one and the query language is SQL3+, an adapted version of SQL3. Started in 1993, the IRO-DB Esprit project [5] has developed a similar approach, but based on the ODMG "standard" [6].

IRO-DB suppor
views of the fe
schemas. IRO-D
through collecti
architecture and

Recently, a
research. This
and an integrati
are Garlic of I
Manifold of A
model data as l
object-relationa
which have bee
we believe that
to support multi
This paper firs
Developed by
application was
support multim
enabled. In the
In the third sec
support Web te
architecture an
the contribution

2. The IRO

IRO-DB is an
operational. It
DBMS : O2. M
features and sor

2.1 Project O

IRO-DB (Inter
project develo
architecture is t
the ODL defini
oriented and re
three layers, th
research center
communication
objects ; the
management. t
processor.

Accordingly
local schemas,

IRO-DB supports an interactive schema integrator workbench to design integrated views of the federated database, and to automatically generate mappings to exported schemas. IRO-DB also focuses on relationship traversal and complex objects handling through collection support. This paper first gives an overview of the IRO-DB system architecture and describes some of the main components of the system.

Recently, a new generation of heterogeneous database systems has appeared in research. This generation typically focuses on a better support of multimedia objects and an integration with web-based technology. Some of the most well known projects are Garlic of IBM Almaden [7], Tsimmis of Standford University [8], Information Manifold of AT&T [9], and Disco from INRIA [10]. While some of these projects model data as labeled graphs with more or less typed nodes (Tsimmis), most still use object-relational (IM) or pure object models (Disco, Garlic). As with relational systems which have been extended to support object technology (e.g., Oracle 8 or Informix), we believe that operational federated object-oriented database systems can be extended to support multimedia objects and Web technology.

This paper first describes the IRO-DB object-oriented federated database system. Developed by a consortium of European partners, IRO-DB is now operational and an application was recently demonstrated. Thus, the difficulty is now to extend it to support multimedia objects, such as geographical data and images, and to make it Web-enabled. In the next section, we survey the IRO-DB architecture and main components. In the third section, we try to isolate the main issues to address for extending it to support Web technology with multimedia database servers. We propose a three-tiered architecture and discuss some query processing issues. In conclusion, we summarize the contributions of this paper and our future plans.

2. The IRO-DB Project

IRO-DB is an object-oriented federated database system. A version is currently operational. It interconnects a relational system INGRES, and three object-oriented DBMS : O2, Matisse and Ontos. In this section, we briefly describe the main system features and some key components.

2.1 Project Overview

IRO-DB (Interoperable Relational and Object-Oriented DataBases) is an ESPRIT project developed in Europe from 1993 to 1996. The novelty of the IRO-DB architecture is to use the ODMG'93 standard [6] as a common object model supporting the ODL definition language and the OQL query language to federate various object-oriented and relational data sources. The IRO-DB architecture is clearly divided into three layers, thus facilitating the cooperative development of the project in several research centers. The local layer adapts local data sources to the ODMG standard ; the communication layer efficiently transfers OQL requests and the resulting collections of objects ; the interoperable layer provides schema integration tools, security management, transaction management, object management, as well as a global query processor.

Accordingly, IRO-DB follows an architecture with three layers of schemas, i.e., with *local schemas*, *import/export schemas*, and *interoperable schemas* (also referred to as

integrated views). A local schema is a local database schema, as usual. An export schema describes in ODMG terms the subset of a database that a local system allows to be accessed by cooperating systems. IRO-DB does not support a global unique integrated schema, but allow application administrators to define *integrated views*. It consists of a set of derived classes with relationships together with mappings to the underlying export schemas.

2.2 System Layer Descriptions

The architecture of the system is organized in three layers of components as represented in figure 1.

At the *interoperable layer*, object definition facilities stand for specifying integrated schemas, which are integrated views of the federated databases. We fully support the ODL view definition language, with many to many relationships. An interactive tool called the *Integrator Workbench (IW)* is offered to help the database administrator in designing integrated views. Views and export schemas are stored in a data dictionary. Object manipulation facilities include an embedding of OQL in the OML/C++ user language and modules to decompose global queries into local ones (global query processor) and to control global transactions (global transaction management). Object definition and manipulation facilities are built upon the integrated object manager (IOM).

The *communication layer* implements object-oriented Remote Data Access (OO RDA) services through the Remote Object Access (ROA) modules, both on clients and servers. Integration of the ROA protocol within the interoperable layer is provided through the object manager, which is able to manipulate collection of any types. Thus, it is possible to invoke OQL/CLI primitives to retrieve collections of objects stored on the local site. OQL/CLI primitives include connection and deconnection, preparation and execution of OQL queries with transfer of results through collections of objects, plus some specific primitives to import at the interoperable layer the exported ODMG schemas, as well as a primitive to perform remote method invocation.

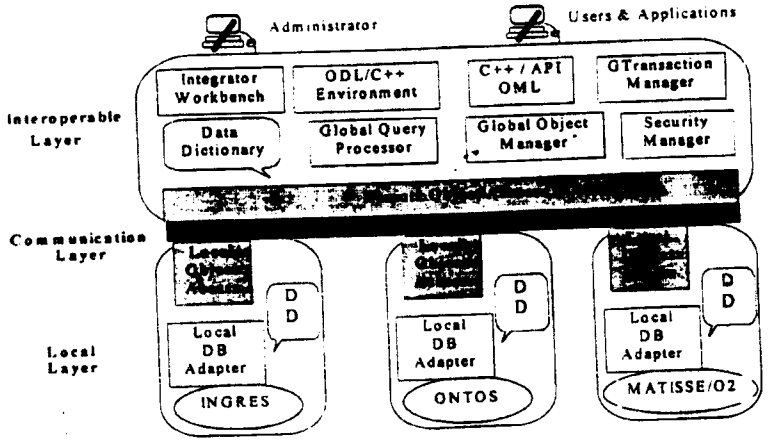


Fig. 1. The IRO-DB system architecture.

RN-01995

The *local layer* is composed of Local Database Adapters (LDA). A local database adapter provides functionalities to make a local system able to answer OQL queries on an abstraction of a local schema in term of ODMG schema (export schema). As an export schema only describes locally implemented types, only locally available functionalities are available for querying through OQL. That means for example that methods and access paths cannot be invoked with simple LDA whose does not know how to export complex ODMG schemas. In that case, only flat objects without relationships are handled at the local layer for relational systems. Of course, if the local system supports the full ODMG model, all syntactically correct ODMG queries are acceptable.

2.3 Main System Components

The Integrator Workbench

The integrator workbench generates from a graphic interface the ODL description of an integrated view, plus some mapping definitions using OQL and C++ method profiles. Integration is done one exported schema after the other. At first, two exported schemas are displayed using a graphical view of the object model. Classes are represented as nodes. Two types of arcs are used to represent relationships and generalizations linking class nodes together. Attributes and methods are directly linked to the class they belong to using aggregation arcs. A naïve schema integration is first perform by union of the two graphs. Then, integration is performed under the direction of the database administrator, which specifies similarities between the export schemas.

For example, with the CIM test-bed application that has been developed to demonstrate the system [11], we integrate two databases as represented in figure 2. Site 1 manages parts and manufacturers. Site 2 manages parts. The G::PART integrated view is a class with a one-to-many relationship referencing manufacturers, i.e., G::MANUF. The integrator workbench generates the view definition in ODL and the mapping to the import schemas given in figure 3.

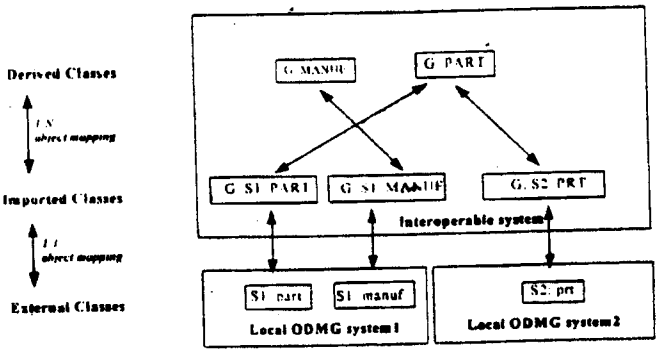


Fig. 2. Example of integrated databases.

```
Interface PART
      (extent  parts
      keys              part_id) {
      attribute String  part_id;
      attribute Date    upd_date;
      attribute         String   description;
      relationship      Set <MANUF> manufs
                          inverse MANUF::parts ; }
Mapping PART {
      origins  sorig, iorig;
      def_ext  select PART(sorig : s_inst, iorig : i_inst)
               from s_inst in s_parts, i_inst in i_prts
               where s_inst.part_id = i_inst.prt_id;
      def_att  part_id as this.sorig.part_id;
      def_att  upd_date as this.sorig.upd_date;
      def_att  description as this.sorig.description;
      def-rel  manufs as
               select me
               from me in manufs
               where ( this.sorig = me.sorig.part) or  (this.iorig = me.iorig.part);
```

Fig. 3. Definition of a derived class.

The Global Query Processor

The *Global Query Processor* goal consists in processing queries against an integrated schema. It is responsible for decomposing the query in order to identify the necessary object transfers and consequently the sub-queries that should be sent to local databases. It is composed of three components described below. Their complete description is presented in [12].

• *Translator*. It manages OQL queries expressed against derived classes and translates them in equivalent OQL queries expressed against imported schemas. The intuitive principle of translating a query expressed on derived classes consists in replacing all derived class extent names by their mapping definition. The translation process uses the derivation specification of each class available in the repository. If there are several layers of derived classes, the query is translated recursively, until it refers to imported classes only.

• *Optimizer*. The optimizer task consists in improving the query processing performance. For that, it applies a set of rules to minimize the object transfers and the communication cost between the IRO-DB client and the local databases. At first, a cost model was designed for the optimizer using a calibrating approach [13]. Although some ODBMSs were calibrated, the cost model was finally not used and heuristics were used. First, rules are applied to flatten the query as much as possible. Next, selections (i.e., restrictions and projections) are moved down the tree. Finally, caching rules are used to avoid transferring objects already in the client cache.

• *Decomposer*. The query decomposer identifies sub-trees which refer to mono-site queries and generates the OQL corresponding sub-queries. The query decomposer

generates an e
executed on lc
query tree to e
The Global Ob
The *Global Ob*
derived objects
interoperable
systems and co
In IRO-DB th
derived class.
of an OQL que
to structured v
other remote d
method.
The Remote O
The Remote C
queries on TC
export schema
CLI interface t
collections of
have been adde
a collection thr
An ad-hoc p
primitives, usir
ROA and unbu
values with a t
collections (list
to the query r
improving the t
The Local Data
The role of loc
the local datab
(LOA). It acce
schema referen
queries. On top
to several SQL
difficult task.
provides an *exp*
database schem
Although we
that each ODB
those of the rel
is mainly imple
into a selection
attributes or me

generates an execution plan composed of two distinct parts : (i) the set of sub-queries executed on local DBMS, (ii) the synthesis query corresponding to the part of the query tree to evaluate globally.

The Global Object Manager

The *Global Object Manager* ensures virtual object management for the imported and derived objects. It allows objects creation and access and guarantees their identity at the interoperable layer. The Object Manager imports objects from external database systems and combines them into a uniform representation by creating surrogate objects. In IRO-DB the management of imported classes differ from the management of derived class. The instantiation of an imported class is not necessary for the evaluation of an OQL query. On the contrary, for all queries that return derived objects as opposed to structured values, it is mandatory to create derived objects since they do not exist in other remote databases. The instantiation of a virtual class is done by its constructor method.

The Remote Object Access

The Remote Object Access component is implementing the CLI interface for OQL queries on TCP/IP. It provides a set of commands to manage contexts, connections, export schemas, statements, results and transactions. We have modified the standard CLI interface to handle OQL queries in place of SQL, but also and mainly to handle collections of objects in place of tuples. Thus, primitives to handle object descriptions have been added. Iterators have been introduced and standard OML primitives to cross a collection through an iterator have been added.

An ad-hoc protocol derived from FAP has been implemented to support the transfer primitives, using XDR for encoding. Commands are bundled into messages by the ROA and unbundled on the server by the LOA. An object is transfered as a structure of values with a tag giving the type of each attribute. Similar objects are organized in collections (list, set, bag and array) both on the server site and the client site, according to the query result type. Collections of similar objects are packed in pages for improving the transfer rate.

The Local Database Adapters

The role of local database adapter (LDA) is to execute received OQL queries on top of the local database and to send back results through the Local Object Access module (LOA). It accesses to the local data dictionary containing the definition of the export schema referenced by the query. Then, it translates the OQL query to a set of local queries. On top of relationnal systems, complex OQL queries are generally translated to several SQL queries. Assembling the results in collections is one of the most difficult task. To populate the dictionary from a local database schema, each LDA provides an *export schema builder*. This is a tool reading the relevant part of the local database schema and building an ODMG counterpart.

Although we design a generic adapter for ODMG databases, we quickly discovered that each ODBMS interface is specific. Thus, except the common libraries which are those of the relational LDA, the object LDAs have no common modules. The O2 LDA is mainly implementing a dynamic OQL on top of O2. Each OQL query is translated into a selection of object identifiers and then into some accesses to the relevant object attributes or methods. The MATISSE LDA is very specific ; it interprets OQL queries

using the MATISSE C API. The ONTOS LDA is quite similar, but with a different target language.

3. Extending IRO-DB to Support Web Multimedia Databases

In this section, we discuss how we plan to extend IRO-DB to the Web, as a mediator supporting object-relational queries on integrated views of multiple data sources, the query being issued from Web browsers. Due to the multimedia nature of the Web, multimedia servers also have to be supported. As an example, we discuss the case of an image server.

3.1 Integrating the Web

Integrating a federated database system to the Web first means accessing it through browsers and Http servers. Web client access is currently dominated by browser software from Netscape and Microsoft (Explorer) providing graphical user interfaces. One major characteristic of such navigators is that their functionality can be easily extended and adapted to specific application needs. There are at least four approaches to extend a browser : (1) **Scripts** written in a script language (e.g., JavaScript) included in Html pages and interpreted by the browser, (2) **Applets** written in Java that are small applications whose intermediate code is included in Html documents to be interpreted by browsers, (3) **Plugs-in** written in any language, compiled and loaded on demand then linked to extensible browsers, (4) **ActiveX controls,** a technology provided by Microsoft, which are persistent modules loaded from the server and integrated to windows browser through the DCOM Microsoft technology.

Among these approaches, it is hard to predict which will become dominant in the future. However, it seems already that scripts are limited to data entry, plugs-in are too heavy to manage, and ActiveX controls are Microsoft dependent. Thus, Java applets seems to be the best technology for integrating IRO-DB functionality with browsers.

On the server side, the interest of DBMS vendors is to provide suitable frameworks to their customers, which allow for the easy creation or adoption of information services for the Web. Currently, there are four approaches to develop Web servers for database applications : (1) **CGI (Common Gateway Interface)** is the mechanism for a user to invoke a program that sends output Html formatted data back to the client. The program can invoke a database manager, e.g., through an ODBC SQL interface or a specific one. For example, the Oracle Web product is based on a CGI interface to process queries and return formatted results. (2) **Server APIs** are specific interface to the server. The Netscape API includes a Database Connectivity Library for direct SQL connectivity to most relational databases, including Oracle, Informix, Sybase and ODBC for DBMS independent connections. The Microsoft API includes dbWeb, an application able to process queries from a Web browser and handles the communication between the browser, an ODBC data source, and a Web server to display the results on an outgoing Web page. (3) **Java-based server APIs** use the Java programming language to create applets on the client side that run programs on the server side. The so-called JDBC (Java Database Connectivity) API proposed by SunSoft makes available a standard interface to any Java applet or application. JDBC is based on the X/OPEN SQL Call Level Interface (CLI), the basis of ODBC. JDBC is

available on
through JDB
variation of
database serv
Visual Basic
Integrating
interoperable
database inte
Web technol
Html files. A
formatted dat
the Web con
Howe'er, JDI
it has to be e
We further de
Finally, fo
architecture, a
the mediator
sent to the m
mediator will
through JDB
multimedia da

3.2 Local Sup

The ODL and
provided by LD
extents and rel
objects ? Let us
documents are
Keywords can b
keywords. Ref
relationships. In
Photographic Ex
types is display.

available on top of ODBC, which means that any relational database is accessible through JDBC. The protocol used to access a relational database from JDBC is some variation of the ISO Remote Data Access (RDA) protocol, depending from the database server. (4) **ActiveX server tools** are pushed by Microsoft. They are written in Visual Basic or Visual C++ and run under the control of ActiveX browser objects.

Integrating the Web also means being able to access multiple data sources from the interoperable layer runtime, hereafter referred to as the *mediator*. A mediator aiming at database integration should support most database servers in an efficient way using Web technology. It should also be able to integrate loosely formatted files, such as Html files. As introduced above, the JDBC API, which can be plugged on any type of formatted data, seems to be currently the best approach for relational data sources in the Web context. There are plans to support object-relational and object databases. However, JDBC does not yet supports multimedia data or semi-structured files. Thus, it has to be extended with specific data types, as images, texts, Html documents, etc. We further detail this point in the communication layer section.

Finally, for mediating between various databases on the Web, a three-tiered architecture, as represented in figure 4, seems to be well suited. The user will contact the mediator home page. Through Java applets, global queries will be formulated and sent to the mediator using HTTP-CGI or Java to Java protocols (e.g., RMI). The mediator will then decompose the query in local sub-queries sent to local sources through JDBC for formatted databases, and through some extensions of it for multimedia databases.

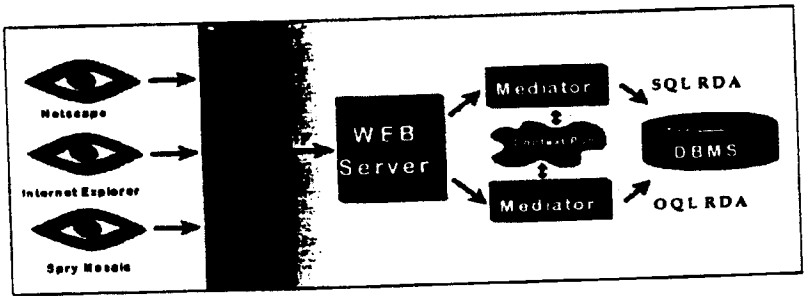


Fig. 4. A three-tiered mediator architecture.

3.2 Local Support for multimedia sources

The ODL and OQL languages are the local description and manipulation languages provided by LDAs in IRO-DB. ODL is similar to CORBA IDL, but with the notion of extents and relationships. Is that sufficient to describe and manipulate multimedia objects ? Let us consider for example an Html file with references to images. Html documents are generally loosely formatted. They are often retrieved using keywords. Keywords can be modeled by a method returning a ranked array of, for example, 10 keywords. References (i.e., HREF) to images can be modeled through 1-1 relationships. Images are generally in Graphical Interchange Format (GIF) or in Joint Photographic Experts Group (JPEG) format. The only available function on such data types is display. Thus, in that case, ODL is sufficient to describe an image.

However, in the case of an image DBMS, content-based queries are possible. In many image systems, pick lists are available for query support. Pick lists gives values for typical features, such as colors, textures, shapes, etc. [14]. Other operations, such as rotate, clip (to select an image region), overlay to check if two images intersect, are possible. Thus, a description of an image can be specified using ODL as given in figure 5.

Query processing is more difficult, as queries in multimedia servers are not exact match queries. Often, a search expression involves uncertainty and fuzziness, with comparison operators like "SIMILAR TO" in conditions. Such predicates have to be defined as operations at the type level in ODL. Sophisticated techniques have been developed for similarity measurement ; they generally compute a measure from 0 to 1. Returning this measure to the mediator is not sufficient as the query processor does not know how to interpret and combine such numbers. Visual query languages as QBIC supports such fuzziness [7], but mediators do not. This is an open problem.

```
Interface image
{ attribute
photoid int ;
content array[1024,1024] int ;
operation
array[10] picklist(image) ;
image rotate (image, angle) ;
image clip (region) ;
boolean overlay (image, image) ;
integer similar (image, image) ; }
```

Fig. 5. A simplified definition of an image database.

A typical query on a database federating employees with their pictures defined as images is then :

```
SELECT Clip(Rotate(I,90),"area")
FROM Employees E, I in E.Pictures
WHERE E.age() > 50 and I.similar("Bandit") > 0.8 ;
```

It retrieves all employees older than 50, which look like the bandit picture. Each resulting picture is rotated of 90° and cut using the clip function to fit in the screen. Processing such a query assuming that employees are handled by some objet-relational database server and that images are managed by a multimedia server is a difficult task that we further investigate below.

3.3 Extending the Communication Layer

The IRO-DB communication layer provides facility to communicate from the mediator to the local adapters. With Java, JDBC can be used to query and update relational databases as explained above. However, JDBC has several shortcomings. First, it is based on SQL and does not support objects yet. An OQL version seems to be in

preparation. Second, it does not support global transactions. Integration to transactional monitors is hard to develop. Thus, we plan to replace the IRO-DB communication layer by JDBC to which some extensions will be incorporated. For multimedia servers, it is not obvious that JDBC will be the right choice. We might then use specific communication protocols based on RMI for example, the Java remote method invocation protocol.

3.4 Extending the Interoperable Layer

The IRO-DB interoperable layer includes design help components and query processing components. Both have to be extended as briefly indicated below.

3.4.1 The Integrator Workbench

The Integrator Workbench should support new data types and new mapping functions from one data type to another. For example, integrating two sets of images, one in GIF, the other in JPEG, requires the knowledge of this data types and of a conversion function from JPEG to GIF. This seems to be feasible by integrating libraries of components in the Workbench. More difficult will be the integration of semi-structured data types, such as Html files. Semi-structured type templates should be abstracted from files and integrated in the workbench. The less specific will be the type templates, the less integration will be feasible.

3.4.2 The Query Processor

In IRO-DB, the query processor does not integrate cost-based optimization as stated above. It just applies simple heuristics as push selection first. This strategy could give bad results in a multimedia context. For example, using the query given in the example above, the optimizer might generate two sub-queries, respectively sent to the employee database and to the picture database, as follows :

```
(Q1) SELECT E.Images
     FROM Employees E
     WHERE E.age() > 50
(Q2) SELECT I, res = Clip(Rotate(I,90),"area")
     FROM I in Pictures
     WHERE I.similar("Bandit") > 0.8 ;
```

and a synthesis query run on the mediator :

```
(Q3) SELECT res
     FROM E in Q1 E, I in Q2
     WHERE I = E.Images.
```

This will be a very bad plan, as every image will be compared to the bandit on the multimedia site and many will be transferred on the net. Avoiding bad plans clearly requires clever heuristics or even a full object cost model [15]. Some operations (e.g., clip) are only possible on certain sites. Thus, algorithms that use the source descriptions to generate executable query plans are required [16]. Also, transforming function expressions and knowing operation costs is not an obvious task. A general framework for this problem is proposed in [17].

3.4.3 *The Object Manager*

The IRO-DB object manager only handles the basic data types of OQL. This is insufficient on the Web, with multimedia objects. A question is : should it handle all data types imported from local databases ? If yes, the object manager has to be extensible. Every manipulated data type should be integrated at the level of the object manager. Thus, when exporting a new type, the LDA will have to supply it to the object manager of the mediator. Except if a portable and transferable language is used everywhere (e.g., Java), this is impracticable. One (poor) solution is to restrict the set of available data types. For example, BLOBs could be used to transfer images. Another (rich) solution is to develop all abstract data types in Java and to download the bytecode where needed. Sending methods should then be taken into account by the query optimizer, as sending bytecode might be costly.

4. Conclusion

In this paper, we describe the IRO-DB federated object-oriented database system architecture. IRO-DB federates relational and object-oriented databases through an object-oriented data model with an associated object query language derived from the ODMG proposal. The overall aim of the project — the provision of building blocks for federated database management — has been recognized worldwide as one of the most important challenges in database research and development at the beginning of the 90's. The project develops suitable object-oriented generalizations of the SQL Access Group protocols extended with object-oriented features for the exchange of complex objects. It takes into account relevant industrial data interchange standards or proposals, such as the OMG architecture and the ODMG model. IRO-DB is a joint effort in Europe, which integrates components developed by various partners. The system is currently operational and a demonstrator CIM application is available.

Our goal is now to extend IRO-DB to make it Web-enabled and to support multimedia data sources. Through this paper, we isolate some of the difficulties to do so. It appears that rewriting the system in Java and using Java packages to support multimedia types and to access databases will help solving many problems. However, it is insufficient. As demonstrated, query optimization has to be re-considered. User interface objects have to be developed as applets and template data types have to be plug in to support semi-structured files. Further, the management of metadata has to be improved to support a large number of data sources, with quick structural changes. However, we do not believe that developing a new data model is necessary as the ODMG data model is sufficient to support references as relationships and complex objects as collections. SQL3 provides the same features. We hope that these two models will converge to a unique one sufficient for most Intranet/Internet federated database applications.

5. Acknowledgments

The authors would like to thank all the IRO-DB participants, and specially Béatrice Finance and Peter Fankhauser for providing good basis for this paper.

6. Referen

1. Bukhres O.,
Hall, 1995.
2. Bouzeghoub
London, 19
3. Sheth and
Heterogenec
September 1
4. Rafi, P.DeS
seamless int
Vienna, Aus
5. Gardarin G.
"IRO-DB
Bukhres an
September.
6. Catell Ed.,
7. Carey et. al
RIDE Work
8. Garcia-Moli
J.,"Integratir
Proceedings
March 1995
9. Kirk T., L
Proceedings
Heterogeneo
10. Tomasic A.,
Design of D
Systems, Ho
11. Ramfos A., I
Manufacturir
95, Vienna,
12. Finance B., F
and Object-
Science, Vol
13. Gardarin G.,
an Object-Or
VLDB'96, M
14. Khoshafian
Kaufman, Sa
15. Gardarin G.,
21st VLDB
16. Levy A., Ra
Source Descr
Bases, pp. 25
17. Naacke H.,
Heterogeneou
BD3, INRIA

6. References

1. Bukhres O.A., Elmagarmid A.K. Ed., "Object-Oriented MultiBase Systems", Book, Prentice Hall, 1995.
2. Bouzeghoub M., Gardarin G., Valduriez P., "Object Technology", Book, Thomson Press, London, 1997.
3. Sheth and J.A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, 22(3):183--236, September 1990.
4. Rafi, P.DeSmedt, W. Kent, M. Ketabchi, W. Litwin, M-C Shan. "Pegasus : A system for seamless integration of heterogeneous information sources", IMS International Conference, Vienna, Austria, 1991.
5. Gardarin G., Gannouni S., Finance B., Fankhauser P., Klas W., Pastre D., and Legoff R., "IRO-DB : A Distributed System Federating Object and Relational Databases", In O. Bukhres and A. Elmagarmid Editors, Object-oriented Multibase Systems, Prentice Hall, September, 1995.
6. Catell Ed., "Object Databases: The ODMG-93 Standard", Book, Morgan & Kaufman, 1993.
7. Carey et. al., Towards Heterogeneous Multimedia Information Systems, In Proc. of the RIDE Workshop on Research Issues in Data Engineering, pp. , Taipeh, March 1995.
8. Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J., and Widom J.,"Integrating and Accessing Heterogeneous Information Sources in TSIMMIS." In Proceedings of the AAAI Symposium on Information Gathering, pp. 61-64, Stanford, March 1995.
9. Kirk T., Levy Alon Y., Sagiv Y. and Srivastava D., "The Information Manifold", Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments, Stanford, March, 1995.
10. Tomasic A., Rashid L., Valduriez P., Scaling Heterogeneous Distributed Databases and the Design of DISCO," in the Proceedings of the 16th Intl. Conf. on Distributed Computing Systems, Hong Kong, 1995.
11. Ramfos A., Fessy J., Finance B., Smahi, V., "IRO-DB : a solution for Computer Integrated Manufacturing Applications", 3rd Intl. Conf. on Cooperative Information System, CoopIS-95, Vienna, Austria, Pp. 125-136, May, 1995.
12. Finance B., Fessy J., Smahi V., "Query Processing in IRO-DB", 4th Intl. Conf. on Deductive and Object-Oriented Databases, DOOD'95, Proceedings, Lecture Notes in Computer Science, Vol. 1013, Springer, Singapore, Pp. 299-318, December, 1995.
13. Gardarin G., Sha F., Tang, Z.H., "Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federate Database System", 22nd Intl. Conf. on Very Large Data Bases, VLDB'96, Mumbai (Bombay), India., Morgan Kaufmann, Pp. 378-389, September, 1996.
14. Khoshafian S., Brad Baker A., "MultiMedia and Imaging Databases", Book, Morgan & Kaufman, San Fransisco, 1996.
15. Gardarin G., Gruser O., Z. Tang, "A Cost Model for Clustered Object-Oriented Databases", 21st VLDB Conference, pp. 323-334, Zurich, Switzerland Sept. 95.
16. Levy A., Rajaraman A., Ordille J., "Querying Heterogeneous Information Sources Using Source Descriptions", in Proc. Of the 22nd International Conference on Very Large Data Bases, pp. 251-262, Mumbai, India, Sept. 1996.
17. Naacke H., Gardarin G., Tomasic A., "Leveraging Mediator Cost Models with Heterogeneous Data Sources", in Proc. French National Workshop on Advanced Databases, BD3, INRIA Ed., Grenoble, September 1997.

Abdelkader Hameurlain A Min Tjoa (Eds.)

Database and Expert Systems Applications

8th International Conference, DEXA '97
Toulouse, France, September 1-5, 1997
Proceedings

Springer

ITEM -12-

A Streaming Media JukeBox - Part II: Netscape

Developer News

Apple Searches for a Few Good Clusters

Search Engines: What's the Difference?

Microsoft, SAP Partnership Deepens

This is the second installment in our series on streaming media. In Column51, A Streaming Media JukeBox, we showed you how to program A streaming media jukebox for Internet Explorer. This week, we'll teach you how to program a streaming media jukebox for Netscape Navigator. The advantage of streaming media over local media is that your users don't have to download and store very large media files on their client machines. Instead, you store the media files on a server and have your users buffer them directly into their viewers, without ever storing a piece of the information. We mentioned in our first installment several applications that currently use streaming media: video on demand, Internet-based training, and on-line classes. Another application we have noticed lately is film promotion by Hollywood studios, showing clips from their newly-released movies on the Internet.

The Media Player comes in two formats: an ActiveX control and a Plugin. The ActiveX control can be used only with Microsoft's Internet Explorer running on the Windows operating system. The Media Player's Plugin can be used with Netscape Navigator as well as Internet Explorer running on different operating systems than Windows. In Column 51 we showed you how to embed Microsoft's Windows Media Player in your page and how to control it with JavaScript. Embedding and controlling the Media Player's Plugin is very different, both in how you embed it in your page and in how you control it from your script. In this column we'll show you how to overcome some deficiencies in controlling the Media Player's Plugin so you will be able to create A Streaming Media JukeBox with similar capabilities to the ActiveX control version. Go ahead and play with the jukebox. The Windows Media Player's Plugin is supported only by Netscape Navigator 4.0 and up. In this part we present the PC version only. We'll present the Mac version in a later column. Please refer to our previous part to learn about the difference between audio, ASF, and ASX file formats

In this column, you'll find out:

A Streaming Media JukeBox - Part II: Netscape

Developer News

Apple Searches for a Few Good Clusters

Search Engines: What's the Difference?

Microsoft, SAP Partnership Deepens

The JukeBox Script

Our streaming media jukebox for the plugin version is very similar to our ActiveX control version from Column 5

```
var streams = new Array();
```

We then define the object constructor as:

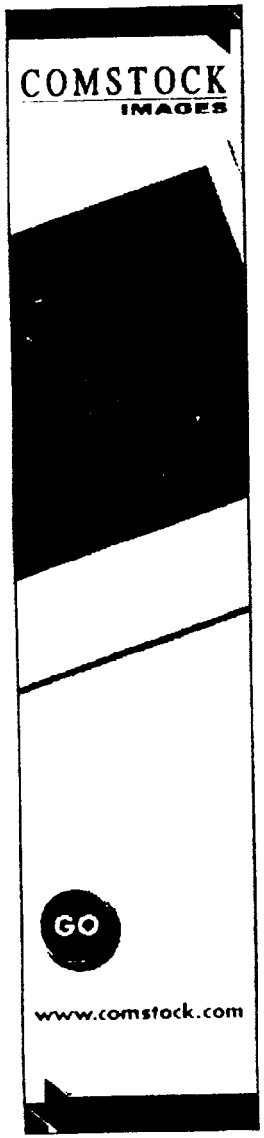
```
function makeStream(url, name) {
  this.url = url;
  this.name = name;
}
```

And finally, we populate the array with several ASX and ASF files:

```
streams[0] = new makeStream("http://msdn.microsoft.com/downloads/samples/Internet/imedia/ne
streams[1] = new makeStream("http://www.advintsol.com/webmaster030399/webmaster.asx", "Micr
streams[2] = new makeStream("mms://netshow.microsoft.com/ms/sbnasfs/wmt/turtle28.asf", "The
streams[3] = new makeStream("mms://netshow.microsoft.com/ms/sbnasfs/wmt/wmt.asf", "Flying W
```

Once we know the number of elements in the array, we can dynamically write a pull-down list. The extra comple window. We first extract the URL parameter and then check with which entry we have a match:

```
function getSelectedEntry() {
```




```
  var query = location.search.substring(2, location.search.length-1);
  for (var i = 0; i < streams.length; i++) {
    if (streams[i].url == query) return i;
  }
  return 0;
}
```

And then we write the pull-down selection list:

```
document.writeln('<SELECT NAME="streams" onChange="change()">');
  var menuSelection = getSelectedEntry();
  for (var i = 0; i < streams.length; i++) {
    if (i == menuSelection) document.writeln('<OPTION VALUE="', streams[i].url, '" SELECTED
    else document.writeln('<OPTION VALUE="', streams[i].url, '">', streams[i].name)
  }
  document.writeln('</SELECT>');
```

The jukebox event handler kicks in whenever there is a change in the selection:

```
function change() {
  var list = document.playerCtrl.streams;
  var streamURL = list.options[list.selectedIndex].value;
  menuSelection = list.selectedIndex;
  document.mediaPlayer.Stop();
  document.displayMode.playOrPause.value = " Pause ";
  location.href = "demo.html" + "?" + '"' + streamURL + '"';
}
```

We first extract the URL of the currently selected item. We then call the Media Player's `Stop()` method and then

RN-02232

http://www.internet.com

Produced by Yehuda Shiran and Tomer Shiran

JupiterWeb networks:

internet.com | EARTHWEB | dev | ClickZ

Search JupiterWeb: Find

Jupitermedia Corporation has four divisions:
JupiterWeb, JupiterResearch, JupiterEvents and JupiterImages

Jupitermedia Corporate Info | Newsletters | Tech Jobs | E-mail Offers

The latest from WebReference.com

Browse > Site Contents

How to Create a JavaScript Windows Interface · Tapestry in Action: Implementing a Tapestry Application · Book Review: Texturing: Concepts and Techniques

Sitemap · Experts · Tools · Services · Email a Colleague · Contact

FREE Newsletters > enter e-mail

The latest from internet.com

Drafting Blueprints for the Digital Home · Plan and Project Your Wireless LAN in 3-D · Broadband on the TV Airwaves

Created: November 9, 1999
Revised: November 23, 1999

URL: http://www.webreference.com/js/column52/jukebox.html

RN-02233

t II: Netscape

[edia Player Plugin's Methods

ı supports several methods. We've built a demo jukebox to show their usage. W
all, Normal, and Large. Here is the HTML form:

```
' Pause " NAME="playOrPause" onClick="handlePlayOrPauseClick()" S
'  Hide Controls " NAME="controls" onClick="handleControlsOnOffCl
' Small " NAME="small" onClick="changeSize(1)" STYLE="font-family
' Normal " NAME="normal" onClick="changeSize(0)" STYLE="font-fami
' Large  " NAME="large" onClick="changeSize(2)" STYLE="font-famil
```

ay button. Every clicking of this button carries out the proper command and swit
ction is the event handler for this button:

```
lick(){

ediaPlayer.GetPlayState();

ay();
ayOrPause.value = " Pause ";

1) {
ay();
```